

Bachelor's Thesis

**Aufbau einer Schnittstelle zwischen MATLAB und
einer Wetterstation über MODBUS**
**Development of a MATLAB gateway to a hardware
weather station via MODBUS**

verfasst von

Andreas Henneberger

Matr.Nr. 2647351

eingereicht am

**Lehrstuhl für Energiewirtschaft und
Anwendungstechnik
Technische Universität München,**

bei

Prof. Dr. rer. nat. Thomas Hamacher

Betreuer: Dipl.-Ing. Christian Kandler und Dipl.-Ing. Patrick
Wimmer

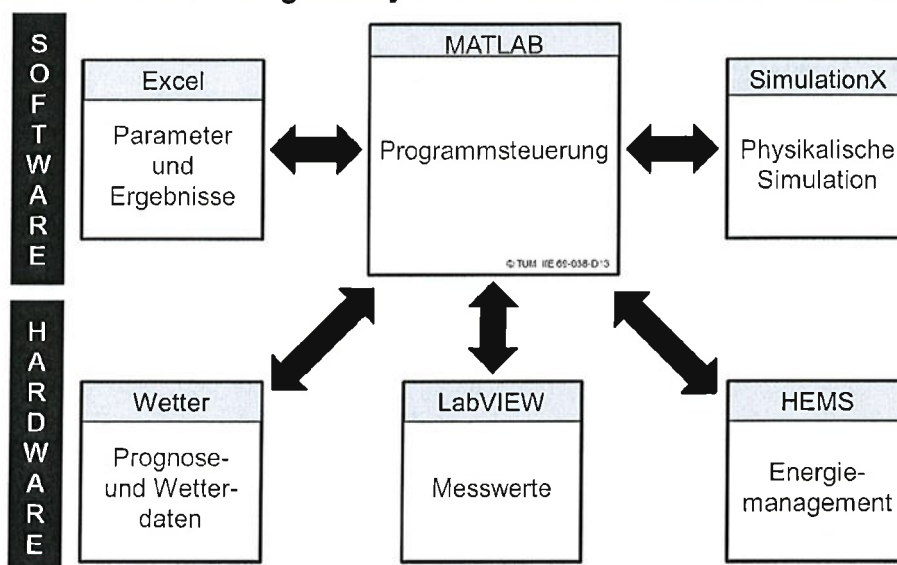
Zusammenfassung

Ziel dieser Arbeit ist es eine Schnittstelle zwischen MATLAB und einer Wetterstation aufzubauen, um darüber Prognosedaten für ein integriertes Energiemanagementsystem bereitzustellen. Diese Informationen sollen dazu dienen die Planungen des Managementsystems im Smart-Micro-Grid hinsichtlich Lastverläufe und Energieerzeugung zu vereinfachen bzw. zu präzisieren. Die Datenbereitstellung erfolgt über einen Langwellenempfänger, dessen Register über eine MODBUS Kommunikation abgerufen werden können. Die meisten gelieferten Werte weisen eine zeitliche Auflösung von 6 Stunden auf. Da das Managementsystem jedoch umso genauer arbeiten kann, je niedriger diese Auflösung ist, ist es mit Aufgabe der Schnittstelle, die Daten in kleineren Zeitintervallen zur Verfügung zu stellen. Der Datenabruf und die Verarbeitung sollen in anderen MATLAB Programmen zum Einsatz kommen. Es ist daher zweckmäßig den Kommunikationsprozess als MATLAB Funktion mit entsprechenden Übergabeparametern zu implementieren. Um die geforderten Aufgabenziele zu erreichen, wurden die Spezifikationen der Wetterstation und des MODBUS-Protokolls analysiert. Mit den aus der Analyse gewonnen Informationen und den in MATLAB zur Verfügung stehenden Methoden, wurde letztlich das Programm umgesetzt. Wie der Leser am Ende der Arbeit feststellen kann, ergibt ein Vergleich der interpolierten Daten mit genauen Wetteraufzeichnungen der LMU ein differenziertes Bild. ...

Aufgabenstellung
Bachelor's Thesis
von
Herrn HENNEBERGER Andreas
Matr.-Nr. 2647351

Thema:
Aufbau einer Schnittstelle zwischen MATLAB und einer Wetterstation
über MODBUS

Development of a MATLAB gateway to a hardware weather station via MODBUS



Das im Rahmen des Schaufensters Elektromobilität geförderte Forschungsprojekt **e-MOBILie - Energieautarke Elektromobilität im Smart-Micro-Grid** hat es sich zum Ziel gesetzt, in einem integrativen Ansatz elektrische Mobilität mit lokaler regenerativer Stromerzeugung zu verknüpfen.

Um ein handlungsfähiges Energiemanagement im Smart-Home gewährleisten zu können, müssen vorab Prognosen für verschiedene physikalische Größen (Temperatur, Einstrahlung, PV-Erzeugung, Lastverlauf, Preise...) erstellt bzw. von extern bezogen und bewertet werden.

Im Rahmen dieser Arbeit soll dazu eine Kommunikationsschnittstelle in MATLAB erstellt werden, welche via MODBUS die Abfrage einer Hardware-Wettervorhersagestation ermöglicht.

Betreuer: Dipl.-Ing. C. Kandler/ Dipl.-Ing. P. Wimmer Tel.: 089-289-28310

Ausgabedatum: 23.09.2013

Aufgabensteller:

Prof. Dr. rer. nat. T. Hamacher

Rechtserklärung

Hiermit erkläre ich,

Name: Henneberger

Vorname: Andreas Helmut

Mat.Nr.: 2647351

dass ich die beiliegende Bachelor's Thesis zum Thema:

Aufbau einer Schnittstelle zwischen Matlab und einer Wetterstation über MODBUS

selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, sowie alle wörtlichen und sinngemäß übernommenen Stellen in der Arbeit gekennzeichnet und die entsprechenden Quellen angegeben habe.

Vom Lehrstuhl und seinen Mitarbeitern zur Verfügung gestellte Hilfsmittel, wie Modelle oder Programme, sind ebenfalls angegeben. Diese Hilfsmittel sind Eigentum des Lehrstuhls bzw. des jeweiligen Mitarbeiters. Ich werde sie nicht über die vorliegende Arbeit hinaus weiter verwenden oder an Dritte weitergeben.

Einer weiteren Nutzung dieser Arbeit und deren Ergebnisse (auch Programme und Methoden) zu Zwecken der Forschung und Lehre, stimme ich zu.

Ich habe diese Arbeit noch nicht zum Erwerb eines anderen Leistungsnachweises eingereicht.

München,

.....

Andreas Henneberger

Inhaltsverzeichnis

I	Einleitung	7
II	Hauptteil	9
1	Aufbau der Wetterstation	10
1.1	Aufbau der Datenstruktur	10
1.2	Technischer Aufbau der Station	11
1.2.1	Senderauswahl und Stationsaufbau	11
1.2.2	Registereinteilung und Schnittstellenparametrierung	12
2	Das Modbus-Protokoll	14
2.1	Verbindungstypen	14
2.2	Nachrichtenaufbau	15
2.3	Registertypen	15
2.4	Nachrichtenverarbeitung	16
2.5	Funktionscodes	16
3	Aufbau und Dokumentation des Funktionscodes	19
3.1	Geforderte Funktionseigenschaften	19
3.2	Vorbereitende Maßnahmen	22
3.2.1	Zuweisung variabler Inputparameter und Variableninitialisierung	22
3.2.2	Aufbau von Strukturen	23
3.2.3	Überprüfung der Eingabeparameter	24
3.2.4	Verfügbarkeitsprüfung der seriellen Schnittstelle	28
3.3	Aufbau der seriellen Schnittstelle	29
3.4	Abgleich der Wetterregion im Register	29
3.5	Festlegung der Timerparameter und Starten des Timers	30
3.6	Abschicken der MODBUS-Anfragen	32
3.7	Senden und Empfangen	36
3.8	Rx-Datenverarbeitung	37
3.9	Datenverarbeitung	37
3.10	Funktionsaufbau	38
4	Datenanalyse	39

III Zusammenfassung und Ausblick	40
Anhang	41
Anhang	42
A Aufbau der Wetterstation	42
B Das MODBUS Protokoll	43
C Abkürzungsverzeichnis	45

Abbildungsverzeichnis

1.1	Verfügbare Prognosedaten WS-K RTU485 WPAia	11
1.2	Standorte der Langwellensender und des Empfängers	11
2.1	Skizze der Funktionsweise des MODBUS Protokolls	14
2.2	Aufbau einer MODBUS Nachricht	15
3.1	Ablaufplan der Funktion forecast_data	20
3.2	Beispiel für den Funktionsaufruf	21
B.1	Ablaufdiagramm für die MODBUS Nachrichtenüberprüfung	43
B.2	Übersicht der zur Verfügung stehenden Funktionscodes in MODBUS	44

Tabellenverzeichnis

1.1	Einstellungsparameter für den Kommunikationsaufbau und den Wetterbereich . .	12
1.2	Schnittstellenparameterbelegung	13
2.1	Übersicht der Registerarten im MODBUS Protokoll	16
2.2	Aufbau einer lesenden Kommunikation mit einem Coil-Register	17
2.3	Aufbau einer schreibenden Kommunikation mit einem Holding-Register	17
2.4	Aufbau einer lesenden Kommunikation mit einem Holding-Register	17
3.1	Datenverlust bei kleinem Updateintervall am Beispiel minimaler Temperatur und Luftdruck	38
A.1	Detaillierte Datenstruktur der Wetterstation[1, S. 17-26]	42

Teil I

Einleitung

In Zukunft wird die Mobilität durch Elektroautos mit geprägt sein. Damit Deutschland auf diesem Technologiefeld eine Spitzenposition einnehmen kann, wurde von der Bundesregierung die Nationale Plattform Elektromobilität initiiert. Ziel dieser Institution ist es Deutschland bis zum Jahr 2020 zum Leitmarkt und Leitanbieter zu entwickeln. Marktvorbereitung, Markthochlauf und der Massenmarkt sind dabei die zu durchlaufenden Phasen. In der Marktvorbereitungsphase, in der wir uns zur Zeit befinden, werden die Ergebnisse aus Forschung und Entwicklung genutzt, um in vier sogenannten Schaufenstern die Modelle und Prognosen für den Markthochlauf zu validieren bzw. bei auftretenden Abweichungen anzupassen.[2] Eines dieser Schaufenster, genannt "Elektromobilität verbindet" wird von den Bundesländern Bayern und Sachsen betreut und finanziert. Das Schaufenster ist aufgegliedert in vier Teilprojekte von denen eines sich den Energiesystemen widmet. Das Themengebiet Energiesysteme ist wiederum in 9 Aufgabengebiete unterteilt, wovon sich eines mit der Integration der Elektromobilität in die dezentrale regenerative Energieversorgung beschäftigt. Ein Aufgabenschwerpunkt hierbei ist es ein integriertes Energiemanagementsystem mittels Aufbau und Betrieb eines Hardware-in-the-Loop Prüfstands zu evaluieren. Da das Energiemanagementsystem auch Vorausschautechnologien einbinden soll, ist es erforderlich Prognosedaten zu erheben.[3]

Teil II

Hauptteil

Kapitel 1

Aufbau der Wetterstation

1.1 Aufbau der Datenstruktur

An dieser Stelle der Bachelorarbeit sollen die grundlegenden Eigenschaften der verwendeten Wetterstation dargestellt werden. Eine gute Kenntnis der Datenstruktur sowie der Datenbereitstellung sind eine zwingende Voraussetzung für den späteren Aufbau der MATLAB Funktion. Die nachfolgenden Angaben in diesem Kapitel können der Hardware Spezifikation entnommen werden [?]. Der Hersteller bietet für die Hardware eine Reihe von Lizenzmodellen an, die den Empfang der Datenmenge bestimmt. Das in dieser Arbeit zum Einsatz kommende Modell nennt sich "WS-K RTU485 WPAia T" und beinhaltet das Prognosepaket "Premium All inclusive advanced", welches es ermöglicht, das komplette Spektrum an Prognosedaten abzurufen. Welche Wetterinformationen genau zur Verfügung stehen, kann der unten aufgeführten Grafik entnommen werden. Für welche Bereiche diese meteorologischen Daten zutreffen muss in der Wetterregion spezifiziert werden. Hier besteht die Möglichkeit für über 1000 Städte in fast ganz Europa die Wetterprognosen abzufragen [1, S. 27-38]. Wie schon in der Einleitung erwähnt, wäre eine niedrige zeitliche Auflösung der Daten wünschenswert, damit das Energiemanagementsystem ohne große Verwerfungen planen kann. Jedoch liegen die meisten Daten in einer Auflösung von 6 Stunden vor, d.h. für ein Intervall von morgens, mittags, nachmittags und abends. Lediglich die mittlere Lufttemperatur wird in einer 1 stündigen Auflösung bereitgestellt. Dieser Umstand wird später im Programmablauf gesondert berücksichtigt. Ein Update der Daten erfolgt ebenfalls alle 6 Stunden. Neben der Auflösung unterscheidet sich auch der Prognosehorizont innerhalb der zugänglichen Daten. Die Spanne reicht von einem bis zu drei Folgetagen. Für den aktuellen Tag, liegen für alle Bereiche Daten vor. Welche meteorologische Ausprägung welche Eigenschaften besitzt, kann in der Tabelle **A.1** im Anhang nachvollzogen werden.

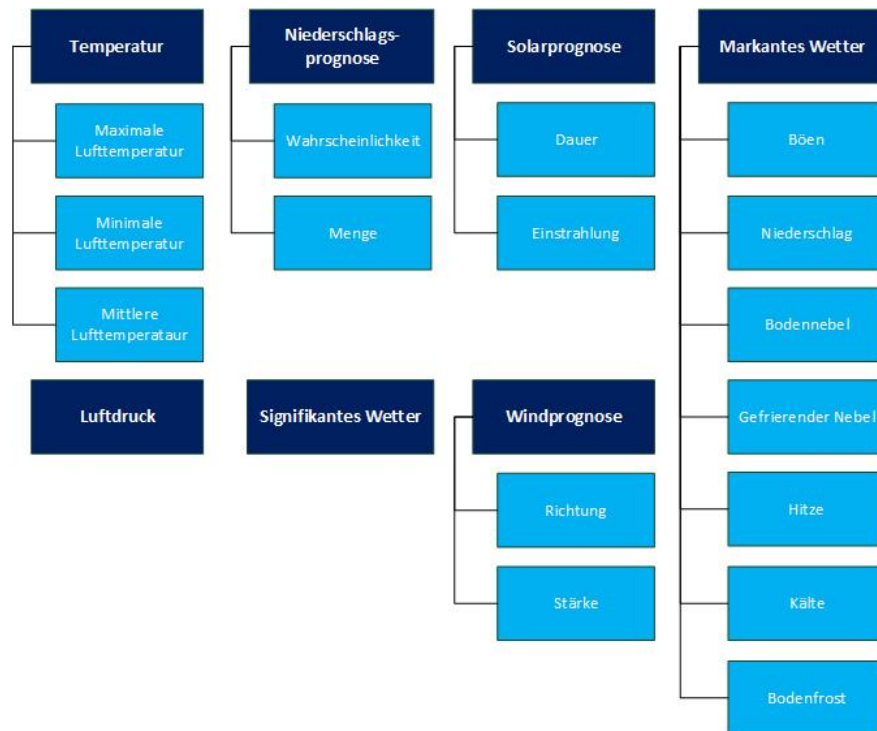


Abbildung 1.1: Verfügbare Prognosedaten WS-K RTU485 WPAia

1.2 Technischer Aufbau der Station

1.2.1 Senderauswahl und Stationsaufbau

Die eingesetzte Wetterstation erhält ihre Daten via Langwelle von drei auswählbaren Sendern:

- Sender Mainflingen DCF 49
- Sender Burg DCF 39
- Sender Lakihegy HGA 22 (Ungarn)

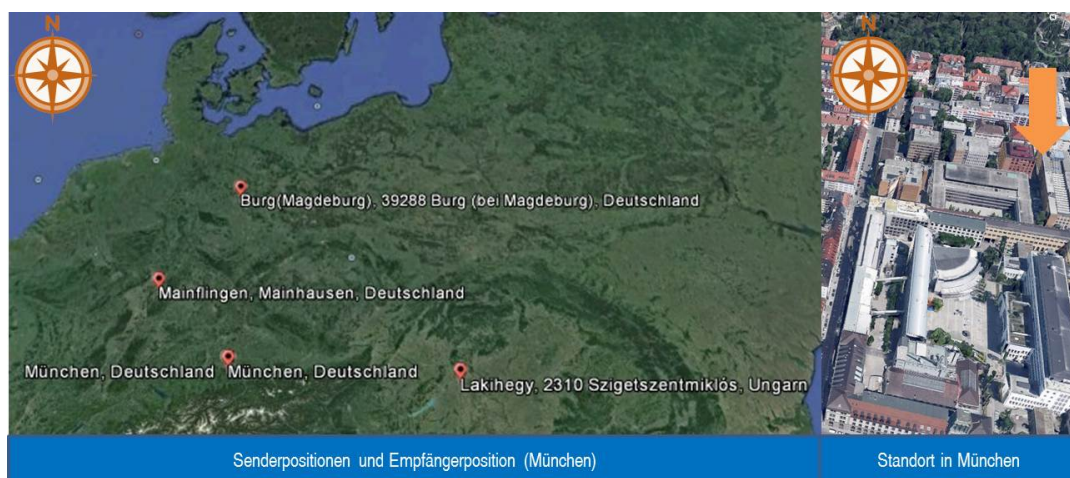


Abbildung 1.2: Standorte der Langwellensender und des Empfängers

Tabelle 1.1: Einstellungsparameter für den Kommunikationsaufbau und den Wetterbereich

Register-adresse	Bezug	Zugriff	Datentyp	Bereich	Bemerkung
110	Senderstation	Lesen/Schreiben	unsigned	0,1,2	0 = DCF 49 1 = HGA 22 2 = DCF 39
111	Empfangsqualität	Lesen	unsigned	0...9	9 ist höchste Qualität
112	Stadt ID	Lesen/Schreiben	unsigned	0...1022	
100	Sekunde (Funkuhr)	Lesen	unsigned		UTC
101	Minute (Funkuhr)	Lesen	unsigned		UTC
102	Stunde (Funkuhr)	Lesen	unsigned		UTC
103	Tag (Funkuhr)	Lesen	unsigned		UTC
104	Monat (Funkuhr)	Lesen	unsigned		UTC
105	Jahr (Funkuhr)	Lesen	unsigned		UTC

Die Wetterstation soll in München aufgebaut werden. Um einen guten Empfang gewährleisten zu können, muss sie entsprechend ausgerichtet werden. Der Hersteller gibt hierzu Kriterien vor, die beachtet werden sollten:

- senkrechter Aufbau des Gehäuses mit nach unten austretendem Kabelstrang
- für einen Innenaufbau in der Nähe zum Fenster
- Mindestabstand von 30 cm zu Metallkonstruktionen oder -flächen
- ausreichende Entfernung zu Geräten die elektromagnetisch abstrahlen
- keine direkte Sonnenbestrahlung für das Einbinden der lokalen Temperatur
- ausreichender Bodenabstand, um Einschneien zu vermeiden
- Ausrichtung zum geografisch günstigsten Sender

Unter Berücksichtigung dieser Empfehlungen wurde die Station in Fensternähe in nordwestlicher Richtung aufgebaut und die Sendestation Mainflingen vorgegeben.

1.2.2 Registereinteilung und Schnittstellenparametrierung

Wie im vorigen Kapitel bereits erläutert, können über das MODBUS Protokoll vier Arten von Registern angesprochen werden. In der Wetterstation sind zwei Register für die Kommunikation vorgesehen. Im Holdingregister können Einstellungsparameter gesetzt und gelesen werden. Eine Übersicht gibt die oben aufgeführte Tabelle **1.1**. Außerdem sind sämtliche meteorologischen Daten in diesem Register abgelegt. Eine Auflistung der den Prognosebereichen zugeordneten Registeradressen gibt die im Anhang befindliche Tabelle A.1. Es ist dabei zu beachten, dass

die Adressen gegenüber den in der Spezifikation des Herstellers Angegebenen, bereits auf die Struktur des Holdingregisters angepasst wurden. D.h. da das Holdingregister mit einer 0 beginnt, wurde von jeder Adresse eine Position abgezogen. Neben dem Holdingregister gibt es noch das Coilregister, in welchem die Zustände für den externen Temperatursensor und die FSK Qualität vorgehalten werden. Die Adressen hierfür sind 1 bzw. 2 und die zugelassenen Werte 1 und 0 geben jeweils den Zustand an. 1 bedeutet der Sensor sowie die FSK Qualität sind in Ordnung. Um eine funktionierende MODBUS Kommunikation über eine serielle Schnittstelle aufbauen zu können, müssen bestimmte Schnittstellenparameter beim Empfänger sowie beim Sender identisch sein. Die Baudrate gibt dabei an, wieviele Bits pro Zeiteinheit übertragen werden können [5, S.169]. Das Paritätsbit zeigt an, ob im übertragenen Byte ein 1 bit Fehler vorliegt. Dies ist dann der Fall, wenn die Zahl der übertragenen Einsen ungerade ist, jedoch das Paritätsbit (= 0 bei even und 1 bei odd) eine gerade Anzahl anzeigt. Das Kommunikationsprotokoll erfordert in diesem Fall eine erneute Übertragung der Information [5, S. 25]. Das Databit gibt lediglich die Anzahl der zu übertragenden bits pro Codewort an. Das Stopbit, entweder 1 oder 2 bits, wird zur Synchronisierung von Sender und Empfänger benötigt, damit diese wissen, wann ein Codewort beginnt und endet [5, S. 169-170]. Per Jumperpositionierung kann die Baudrate von 9600 auf 19200 in der Wetterstation geändert werden, die Werkseinstellung von 19200 wurde aber in dieser Arbeit beibehalten. Ebenso einstellbar über Jumper ist die Slave-Adresse der Wetterstation. Aber auch hier wurde der voreinstellte Wert nicht verändert. Nachdem nun das Modbusprotokoll und die Funktionsweise der Wetterstation bekannt sind, kann mit der Umsetzung der MATLAB Funktion begonnen werden.

Tabelle 1.2: Schnittstellenparameterbelegung

Parameter	Wert
Baudrate	19600
Parität	even
Databit	8
Stopbit	1
Slave-Adresse	03

Kapitel 2

Das Modbus-Protokoll

Nachdem die Wetterstation über MODBUS kommuniziert, soll in diesem Kapitel das MODBUS Protokoll näher erläutert werden. Dabei wird überwiegend Bezug auf die offizielle MODBUS Spezifikation genommen [4]. Angesiedelt auf der ersten, zweiten und siebten Ebene des OSI Modells und damit einfach zu handhaben, ist MODBUS als Kommunikationsprotokoll in der Industrie weit verbreitet. Die unten stehende Abbildung **2.1** dient zur ersten Orientierung der nachfolgend behandelten Themengebiete.

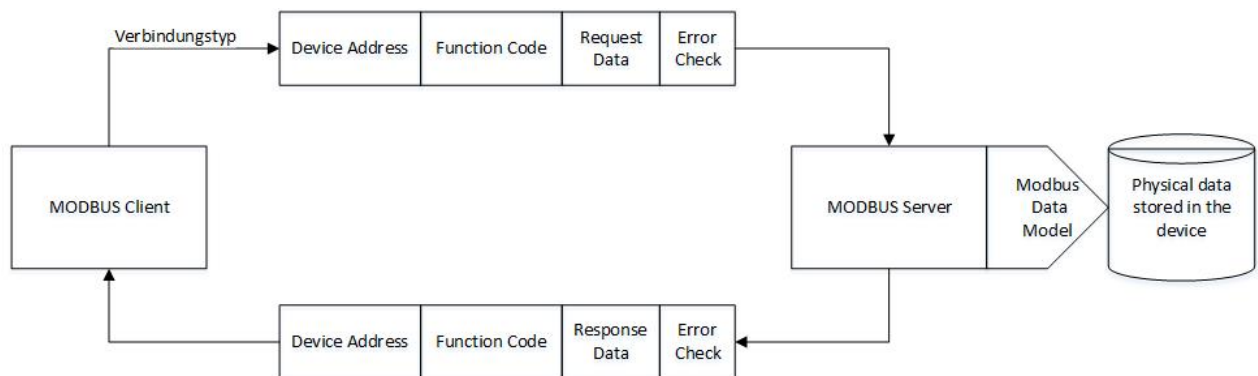


Abbildung 2.1: Skizze der Funktionsweise des MODBUS Protokolls

2.1 Verbindungstypen

Es ist möglich das Protokoll auf drei Verbindungstypen zwischen Client und Server einzusetzen. Dazu zählen:

- eine Internetverbindung TCP/IP
- eine asynchrone serielle Verbindung (z.B. RS-232, RS-422, RS-485, etc.)
- eine MODBUS Plus Verbindung

In dieser Arbeit ist die Wetterstation seriell über eine RS-485 Schnittstelle mit dem Rechner verbunden. Die RS-485 Schnittstelle bietet den Vorteil, dass die Verbindung der Netzwerkteilnehmer wie bei einer RS-232 Schnittstelle nur über eine Zweidrahtleitung erfolgen kann. Jedoch

können im Gegensatz zur RS-232 Schnittstelle bis zu 32 Teilnehmer im Netzwerk angeschlossen werden. Die Netzwerklänge kann ohne Verstärker bis zu 1200m betragen [?]. Diese Eigenschaften bieten sich an, um die Wetterstation in ein Netzwerk zu integrieren, welches von einem Energiemanagementsystem gesteuert wird.

2.2 Nachrichtenaufbau

Wie eine typische MODBUS Nachricht aufgebaut ist, zeigt die unten stehende Abbildung **2.2**. Die PDU ist unabhängig vom Netzwerk auf dem das Protokoll eingesetzt wird und setzt sich

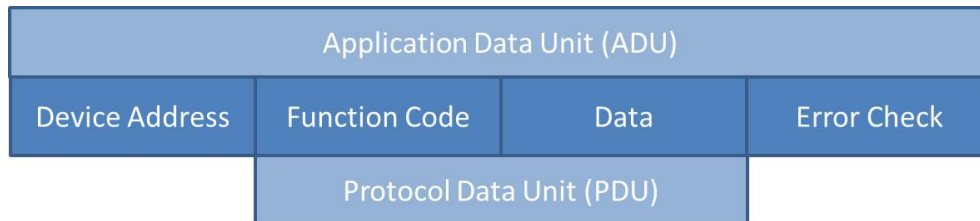


Abbildung 2.2: Aufbau einer MODBUS Nachricht

aus dem Funktionscode und den zu übermittelnden Daten zusammen. Mit einem Byte codiert gibt der Funktionscode an, ob eine schreibende oder lesende Kommunikation an welcher Art Register vorgenommen werden soll. Er kann aber auch einfach nur eine Aktion ausführen. In der Spezifikation werden drei Funktionscodearten genannt, öffentliche, benutzerdefinierte und reservierte Funktionscodes von denen in dieser Arbeit aber nur die Öffentlichen interessieren. Im Falle einer Anfrage des Client, enthält der Datenblock die entsprechenden Informationen über die genaue Adresse und Anzahl der zu lesenden oder beschreibenden Register. Für einen Schreibprozess wird hier auch der notwendige Input angegeben. Die abgefragten Daten des Servers sind ebenfalls im Datenblock untergebracht. Die Device-Adresse und der Error-Check sind Informationen, die für das Netzwerk sprich den Verbindungstyp zwischen den Geräten eine Rolle spielen. Wie eben kurz skizziert, unterscheidet das MODBUS Protokoll zwischen drei Arten von PDUs, die nachfolgend zusammengefasst aufgeführt sind:

- die Anfrage-PDU besteht aus dem Funktionscode und den Anfragedaten
- die Antwort-PDU besteht ebenfalls aus einem Funktionscode und den Antwortdaten
- die Fehler-PDU besteht aus dem Fehlerfunktionscode (Funktionscode + 0x80) und der Fehlermeldung

Die Bytereihenfolge im Datenblock folgt der big-endian Anordnung, d.h. das Most Significant Bit kommt an erster Stelle und das Least Significant Bit an Letzter.

2.3 Registertypen

Die Daten, die vom Client abgefragt werden können, müssen physikalisch im Speicher des Servers liegen. Eine Verknüpfung dieses Speichers mit den zur Verfügung stehenden Registern im

Tabelle 2.1: Übersicht der Registerarten im MODBUS Protokoll

Register	Wortlänge	Zugriff	Info
Diskreter Input	1 bit	Lesen	Daten werden durch ein I/O System bereitgestellt
Coils	1 bit	Lesen/Schreiben	Daten können über ein Anwendungsprogramm geändert werden
Input Register	16 bit	Lesen	Daten werden durch ein I/O System bereitgestellt
Holding Register	16 bit	Lesen/Schreiben	Daten können über ein Anwendungsprogramm geändert werden

MODBUS Protokoll ermöglicht den Zugriff. Es werden vier Registerarten unterschieden, die in der Tabelle **2.1** aufgezeigt sind. Jedes dieser Register besitzt einen Adressraum der bei 0 beginnt und bei 65535 endet.

2.4 Nachrichtenverarbeitung

Die Abbildung **B.1** im Anhang zeigt den Ablauf einer Nachrichtenüberprüfung und an welcher Position welcher Fehlercode gesendet wird, wenn die Nachricht fehlerhaft ist.

2.5 Funktionscodes

Da der Funktionscode ein entscheidender Baustein in der MODBUS Nachricht ist, ist es wichtig die für die Zwecke dieser Arbeit Wichtigen zu identifizieren. Die im Anhang dargestellte Abbildung **B.2** zeigt die zur Verfügung stehenden Funktionscodes. Gelb markiert sind dabei die Codes, die für die Kommunikation zwischen MATLAB und der Wetterstation Bedeutung haben. Wie schon im Abschnitt 1.2.2 auf Seite 13 beschrieben, müssen für den Fall einer Abfrage der Zustände des Temperatursensors oder der FSK Qualität die Coiladressen 0 oder 1 ausgelesen werden. Hierzu reicht es also jeweils eine Adresse in der MODBUS Nachricht anzugeben und die auszulesende Adresszahl auf 1 zu setzen. Die Anfrage- und Antwortnachricht für einen funktionierenden Temperatursensor ist in der Tabelle **2.2** beispielhaft dargestellt. Alle zwei Byte breiten Worte, wie zum Beispiel die Startadresse, setzen sich aus einem sogenannten High-Byte (H-Byte) und einem Low-Byte (L-Byte) zusammen. Ein weiterer wichtiger Funktionscode ist der, mit dem man in das Holding-Register Werte schreiben kann. Wie im Abschnitt 1.2.2 auf Seite 12 in der Tabelle 1.1 nachgelesen werden kann, wird die zu beobachtende Wetterregion mit einem Wert im Holdingregister an der Adresse 112 festgelegt. Die hierzu notwendige Kommunikation ist in der Tabelle 2.3 als Beispiel skizziert. Der wohl wichtigste und am meisten verwendete Funktionscode in dieser Arbeit ist der zum Lesen des Holding-Registers. Über ihn werden sämtliche Prognosedaten ausgelesen. Auch hier soll ein Beispiel in der Tabelle 2.4 den Aufbau verdeutlichen. In dem gezeigten Beispiel werden alle Wetterdaten, insgesamt 96 Werte, für die Mittlere Temperaturprognose abgerufen. Schlägt ein Kommunikationsprozess fehl,

Tabelle 2.2: Aufbau einer lesenden Kommunikation mit einem Coil-Register

Nachrichtentyp	Nachrichtenteil	Wortlänge	Inhalt
Anfrage	Funktionscode	1 Byte	0x01
	Startadresse	2 Bytes	H-Byte 0x00 L-Byte 0x00 (0x0000 bis 0xFFFF möglich)
	Adressanzahl	2 Bytes	H-Byte 0x00 L-Byte 0x01 (1 bis 2000 (0x7D0) möglich)
	Funktionscode	1 Byte	0x01
	Byteanzahl	1 Byte	1 (Ist das Ergebnis von Adressanzahl mod 8 = 0, so ergibt sich die Byteanzahl aus dem Ergebnis der Adressanzahl dividiert durch 8, andernfalls wird um ein Byte erhöht.)
Antwort	Coil Status	n Bytes	00000001 (8 Coilzustände werden mit einem Byte angezeigt. Das Most Significant Bit im Antwort Byte steht dabei für die höchste Registeradresse.)

Tabelle 2.3: Aufbau einer schreibenden Kommunikation mit einem Holding-Register

Nachrichtentyp	Nachrichtenteil	Wortlänge	Inhalt
Anfrage	Funktionscode	1 Byte	0x06
	Registeradresse	2 Bytes	H-Byte 0x00 L-Byte 0x70 (0x0000 bis 0xFFFF möglich)
	Registerinput	2 Bytes	H-Byte 0x01 L-Byte 0x61 (0x0000 bis 0xFFFF möglich)
Antwort	Funktionscode	1 Byte	0x06
	Registeradresse	2 Byte	H-Byte 0x00 L-Byte 0x70
	Registerinput	2 Byte	H-Byte 0x01 L-Byte 0x61

Tabelle 2.4: Aufbau einer lesenden Kommunikation mit einem Holding-Register

Nachrichtentyp	Nachrichtenteil	Wortlänge	Inhalt
Anfrage	Funktionscode	1 Byte	0x03
	Startadresse	2 Bytes	H-Byte 0x00 L-Byte 0x00 (0x0000 bis 0xFFFF möglich)
	Adressanzahl	2 Bytes	H-Byte 0x00 L-Byte 0x5F (1 bis 125 (0x7D) möglich)
Antwort	Funktionscode	1 Byte	0x03
	Byteanzahl	2 Byte	2 x N (N = Adressanzahl)
	Registeroutput	N x 2 Bytes	

so wird vom Server statt der Antwortnachricht eine Fehlnachricht gesendet. Es sind folgende Fehlnachrichten vorgesehen:

- Code 01 Ungültige Funktion
- Code 02 Ungültige Adressdaten
- Code 03 Ungültige Daten
- Code 04 Fehler beim MODBUS Server

Code 1 kann auftreten, wenn die entsprechende Funktion im Gerät nicht implementiert ist oder der Server sich in einem falschen Zustand befindet. Code 02 wird dann gesendet, wenn in der Anfrage mehr Register ausgelesen werden sollen, als zur Verfügung stehen. Code 03 gibt an, dass es sich bei dem im Datenblock befindlichen Wert um einen für den Server nicht Gültigen handelt. Code 04 wird übermittelt, wenn beim Server während der Bearbeitung der Anfrage ein Fehler aufgetreten ist.

Kapitel 3

Aufbau und Dokumentation des Funktionscodes

3.1 Geforderte Funktionseigenschaften

Die zu schreibende MATLAB Funktion soll nach Fertigstellung in weiteren MATLAB Programmen zum Einsatz kommen. Daher ist es wichtig, dass sämtliche Daten die für das Ausführen erforderlich sind bereits im Code vorliegen und nicht importiert werden müssen. Es sollen auch sonst nach dem Ausführen keine weiteren Maßnahmen oder Eingaben getätigt werden müssen. Um diese Voraussetzungen zu erfüllen müssen vor allem große Datensätze im Code eingebunden werden. Bei dieser Arbeit ist das zum einen die Städteliste mit ihren über 1000 Einträgen und zum anderen das Verzeichnis mit allen Registeradressen, in Summe über 340 Positionen. Daneben gibt es noch ein paar weitere Eigenschaften, die an dieser Stelle kurz aufgeführt werden.

- wiederholte Ausführung des Datenabrufs in bestimmten Zeitabschnitten ohne dabei MATLAB komplett zu blockieren
- Handhabung der kompletten MODBUS Kommunikation, insbesondere des Datenabrufs, der -verarbeitung und der Parametersetzung
- Interpolation der ausgelesenen Werte, um unterschiedliche zeitliche Auflösungen zu erhalten
- Datensicherung derart, dass jeder Datenabruf in einer eigenen Datei und die Summe aller abgerufenen Werte in einer anderen Datei gespeichert werden
- Fehlervermeidung bei der Eingabe von Inputparametern
- Aufbau und Beendigung der seriellen Schnittstelle mit der Wetterstation
- einfache und kurze Inputparameter

Nachdem die Eigenschaften nun bekannt sind, soll in den nachfolgenden Unterkapiteln die genaue Umsetzung im Programmcode erläutert werden. Hierzu wird zunächst eine grobe Struktur des Programmaufbaus in Abbildung **3.1** gegeben. Wie in der Abbildung zu erkennen, ist es möglich

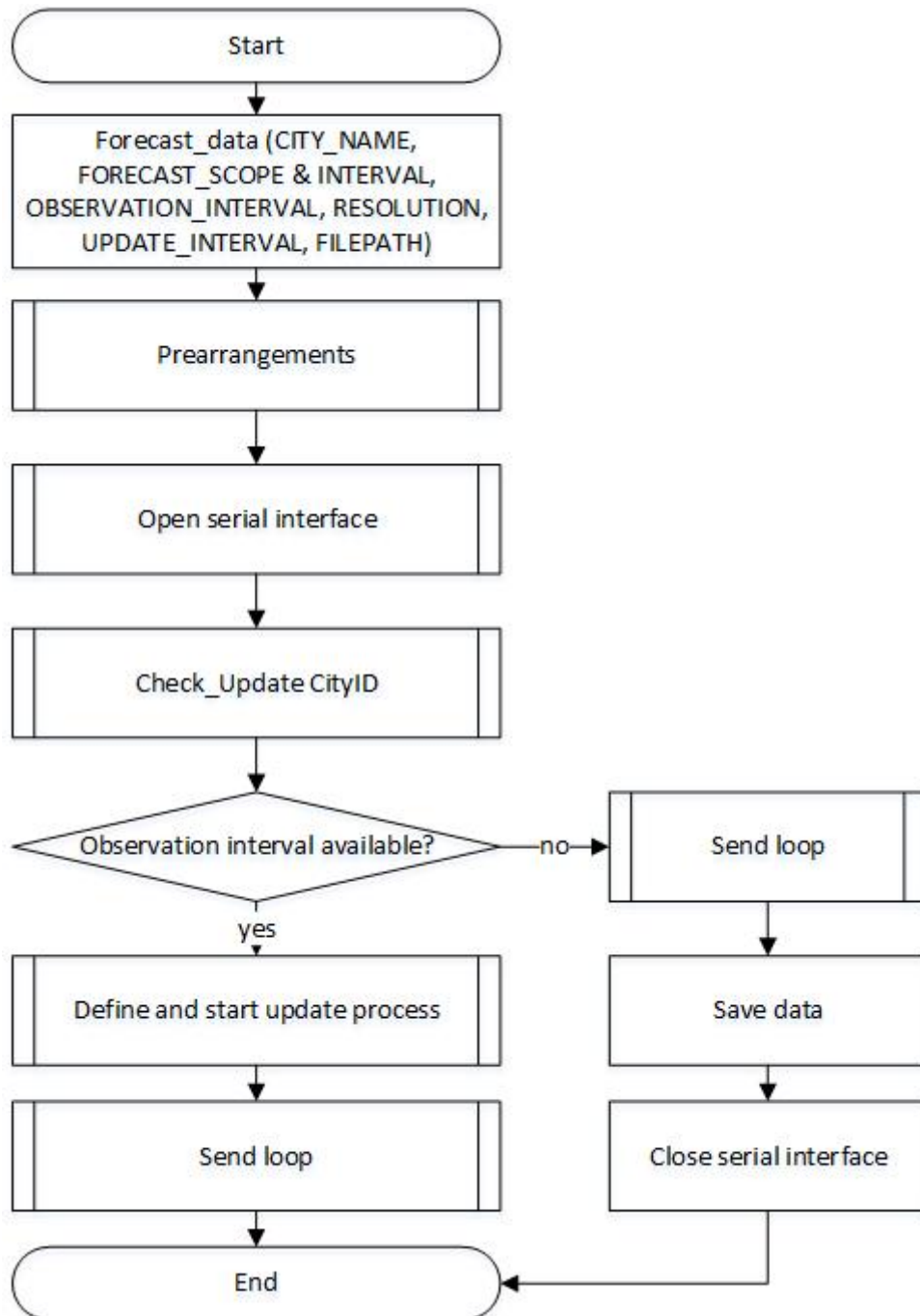


Abbildung 3.1: Ablaufplan der Funktion `forecast_data`

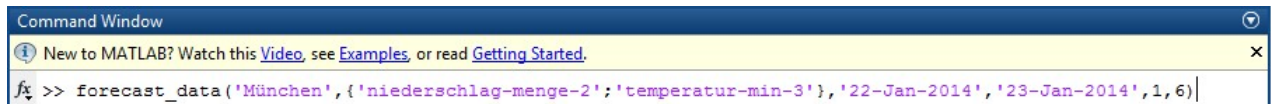


Abbildung 3.2: Beispiel für den Funktionsaufruf

für die Funktion `forecast_data` sieben Eingabeparameter zu definieren.

1. Wetterregion
2. Prognosebereichsdefinition
3. Start des Beobachtungszeitraums
4. Ende des Beobachtungszeitraums
5. zeitliche Auflösung
6. Updateintervall
7. Pfadangabe des Speicherortes (optional)

Die Wetterregion wird als Stadtname in Stringformat übergeben. Die Prognosebereichsdefinition besteht aus drei Teilen, die als String in dieser Format 'Prognosebereich-Prognosedetail-Prognoseintervall' aufgebaut ist. Der erste Teil gibt den Wetterbereich an. Hier stehen alle Einträge in der Spalte Prognosebereich der Tabelle A.1 im Anhang zur Verfügung. Die Prognosedetaildaten können der zweiten Spalte dieser Tabelle entnommen werden. Der dritte Parameter gibt die Anzahl der auszuwertenden Tage an, dabei steht der Wert 1 für den aktuellen Tag ohne Prognose und die Werte 2, 3, 4 für eine entsprechende Erweiterung des aktuellen Tages um die Prognosetage 1, 2 und 3 sofern vorhanden. Möchte man keine gesonderte Auswahl treffen und einfach alle möglichen Daten abrufen, so kann hier an dieser Stelle der Input 'all' erfolgen. Der Beobachtungszeitraum wird durch zwei Daten begrenzt, die ebenfalls als String in dieser Formation 'dd-mmm-yyyy', wobei mit mmm der englische Monatsname gemeint ist, angegeben werden. Die zeitliche Auflösung wird als Double eingetragen. Hier sind die Werte 1, 0.5, 0.25, 0.08 stellvertretend stehend für 1 Stunde, eine halbe Stunde, eine viertel Stunde und 5 Min., möglich. Ebenso als Double wird das Updateintervall definiert. Hier stehen die Werte 6, 12, 24 zur Verfügung, welches einem vier-, zwei- und einmaligen Update am Tag entspricht. Die Pfadangabe wird wiederum als String eingegeben. Die nächste Abbildung gibt hierzu ein Beispiel. Es wird für die Region München die Niederschlagsmenge für den heutigen und ersten Folgetag, die minimale Temperatur für den heutigen und die beiden nachfolgenden Tage, über einen Zeitraum von zwei Tagen abgerufen. Dabei beträgt die zeitliche Auflösung 1 Stunde und das Intervall in denen Updates gestartet werden 6 Stunden. Nachdem hier kein Speicherpfad angegeben ist, wird ein Ordner mit der Bezeichnung „Aufzeichnungen“ im aktuellen MATLAB Ordner erstellt und als Speicherort ausgewählt.

3.2 Vorbereitende Maßnahmen

3.2.1 Zuweisung variabler Inputparameter und Variableninitialisierung

```
if ~isempty(varargin)
    if size(varargin,2) < 4
        fprintf(2,['Bitte geben Sie das Start- und Enddatum des' ...
            'Boabachtungszeitraums\n sowie die Aufloesung und' ...
            'das Updateintervall an.\n'],char(10));
        error('Zu wenig Inputparameter!');
    else
        start_observation      = varargin{1};
        end_observation        = varargin{2};
        resolution              = varargin{3};
        update_interval         = varargin{4};
    end
    if size(varargin,2) > 4
        filepath                = varargin{5};
    else
        if size(pwd,2) < 4
            filepath             = [pwd,'Aufzeichnungen'];
        else
            filepath             = [pwd,'\Aufzeichnungen'];
        end
        [s,mess,messid]         = mkdir(filepath);
        if ~isempty(mess)
            fprintf('Ordner existiert bereits.\n');
        end
    end
else
    if size(pwd,2) < 4
        filepath                = [pwd,'Aufzeichnungen'];
    else
        filepath                = [pwd,'\Aufzeichnungen'];
    end
    [s,mess,messid]             = mkdir(filepath);
    if ~isempty(mess)
        fprintf('Ordner existiert bereits.\n');
    end
    resolution = 1;
end

% If only one forecast definition is requested, convert char input into
% cell array.
if ~iscell(fc_def)
    fc_def = {fc_def};
end

% Set daychange_flag and daychange_counter to 0 for the first execution
daychange_flag                = 0;
```



```

daychange_counter    = 0;
assignin('base','daychange_flag',daychange_flag);
assignin('base','daychange_counter',daychange_counter);

% Initialize or reset with new function call data container
weather_data        = [];
new_data             = [];

% Set device id
device_id            = '03';

```

Für den Fall, dass variable Inputparameter übergeben wurden, wird zuerst geprüft, ob die Anzahl der geforderten Werte vorhanden ist. Ist das nicht der Fall, so wird eine Meldung an den Nutzer ausgegeben und die Funktion beendet. Stimmt die Anzahl, werden die Werte Funktionsvariablen zugeordnet. Ist zudem noch eine Pfadangabe zu einem Speicherort der Funktion übergeben worden, so wird diese ebenfalls einer Funktionsvariablen zugewiesen. Wurde keine Angabe hierzu gemacht, so wird im aktuellen MATLAB Ordner ein Ordner „Aufzeichnungen“ als Speicherplatz definiert. Dabei wird nachgesehen, ob lediglich eine Laufwerksangabe vorliegt oder nicht. Existiert der Ordner bereits, erfolgt eine Meldung auf deutsch. Wurden keine variablen Parameter übergeben, so wird die zeitliche Auflösung auf 1 Stunde festgesetzt und ebenfalls ein Speicherort vorgegeben. Nachdem die *forecast_defintion* sowohl als String als auch als Cell-Array übergeben wird, im späteren Programmablauf aber nur ein Cell-Array erwartet wird, muss noch eine Konvertierung stattfinden. Mit dem ersten Funktionsaufruf, an dem noch kein Tageswechsel auftreten kann, werden die Variablen *daychange_flag* und *daychange_counter* auf Null gesetzt und dem Base-Workspace zugewiesen. Ebenso initialisiert werden die späteren Datencontainer *weather_data* und *new_data*. Wie in Tabelle 1.2 gezeigt, lautet die Slave-ID der Wetterstation „03“. Diese wird hier der *device_id* zugeordnet.

3.2.2 Aufbau von Strukturen

```

% Create a table with all available forecast definitions.
if strcmp(fc_def,'all') == 1
    fc_def = create_table();
end

% Create the structure with all register addresses
data = create_reg_data();

% Create the list with all available city ids
city_list = create_city_list;

% Assign both created structures to base workspace
assignin('base','register_data_hwk_kompakt',data);
assignin('base','city_list',city_list);

```

Wie bereits im vorigen Kapitel 3.1 angekündigt, müssen große Datensätze in Strukturen gepackt werden um später aus ihnen Daten zu gewinnen. Sollen alle Wetterdaten abgerufen werden, ist es erforderlich ein Cell-Array aufzubauen, welches alle Wetterdatenabfragen beinhaltet. Danach werden Strukturen angelegt, die die Registeradressen und Städtenamen abbilden. Die letzten beiden Strukturen müssen wieder im Base-Workspace verfügbar sein da andere Funktionen auf sie zurückgreifen werden.

3.2.3 Überprüfung der Eingabeparameter

```
% Determine the number of requests.
size_table_data      = size(fc_def,1);

% Input check, if all inputs are correct, create data container, else print
% error message.
for z = 1:size_table_data
    [correct_input, error_msg, city_id, longitude, latitude] = ...
        input_check(fc_def{z}, city_name, varargin);
    if true(correct_input)
        weather_data = create_data_struct(fc_def{z}, weather_data, 'weather_data');
        new_data = create_data_struct(fc_def{z}, new_data, 'new_data');
    else
        fprintf(2, '%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n', error_msg{1}, ...
            error_msg{2}, error_msg{3}, error_msg{4}, error_msg{5}, ...
            error_msg{6}, error_msg{7}, error_msg{8}, error_msg{9}, ...
            error_msg{10} , char(10))
        error('Die oben aufgefuehrten Eingabeparameter sind nicht korrekt.');
```

```
    end
end

% Assign data container to base workspace. In the base workspace now the
% city_list, the data container and the register
% address structure are available.
assignin('base', 'weather_data', weather_data);
assignin('base', 'new_data', new_data);
```

Diese for-Schleife bearbeitet alle übergebenen Wetterdatenabfragen, prüft sie auf Gültigkeit und erstellt zugleich die entsprechenden Datencontainer. Die Datencontainer werden im Base-Workspace eingetragen. Sind ein oder mehrere Eingabeparameter falsch, werden diese dem Nutzer mit einer Nachricht angezeigt und die Funktion beendet.

```
function[ val_inpt, err_msg, c_id, lng, lat ] = input_check( fc_def, city, varargin )
```

Die Funktion `input_check` liefert als Outputparameter einen Vektor der angibt, welche Parameter gültig sind. Zusätzlich werden die generierten Fehlermeldungen, die ID der Wetterregion, der Breitengrad und Längengrad ausgegeben.

```

varargin = varargin{:};

% #### Check city ####
% Check for the right spelling and availability of city name
% val_inpt(1) will be 1 for existence and c_id contains the numeric
% city id.
[val_inpt(1), c_id, lng, lat] = get_city_id(city);

% Create failure message for a non existent city name
if val_inpt(1) == 0
    err_msg{1} = ['Diese Stadt kann in der CityList nicht gefunden werden: ' ...
                city];
end

```

Im ersten Abschnitt des Input Checks wird die Funktion `get_city_id` aufgerufen, um die ID der Wetterregion, den Längen- und Breitengrad zu ermitteln. Ist die ID nicht in der Liste zu finden, wird die entsprechende Vektorposition *val_inpt* auf false gesetzt.

```

function [ city_id_correct, c_id, lng, lat ] = get_city_id( cityname )
%Gets the city id from the city list
% Detailed explanation goes here
citylist = evalin('base','city_list');
citylist.CityName = nominal(citylist.CityName);
city_id_dataset = citylist(citylist.CityName == cityname,:);
if isempty(city_id_dataset)
    city_id_correct = 0;
    c_id = '';
    lng = '';
    lat = '';
else
    c_id = city_id_dataset(:,1);
    lng = city_id_dataset(:,3);
    lat = city_id_dataset(:,4);
    city_id_correct = 1;
end
end

```

In dieser Funktion muss zuerst die im Base-Workspace befindliche Variable *city_list* zugänglich gemacht werden. Danach wird die Spalte mit den Städtenamen mit `nominal` konvertiert, um im darauffolgenden Schritt eine einfache Suche der Position in der Liste zu starten, die dem Städtenamen entspricht. In der Variable *city_data_set* sind nun alle Werte dieser Liste, die der Position entsprechen, enthalten. Eine einfache Wenn-Dann-Bedingung weist die Daten den Outputparametern zu.

```

% Check for the right definition of forecast, required scheme
% 'forecast_scope-fc_def-interval'
check = strfind(fc_def,'-');

```

```

% Create failure message if less or more then two '-' are existent in the
% forecast definition string
if size(check,2) ~= 2
    val_inpt(2) = 0;
    err_msg{2} = ['Die Wetterdatenanfrage wurde nicht korrekt definiert: '...
                char(fc_def)];

% If no error put val_inpt(2) to 1
else
    val_inpt(2) = 1;
% Disjoint the forecast definition into separte parts (1) forecast_scope
% (2) forecast_detail (3) forecast_interval
    fc_def = regexp(fc_def, '-', 'split');
% Define the possible values for each part of the forecast definition
    forecast_scope = {'niederschlag', 'wind', 'temperatur', 'solarleistung', ...
                    'markantes.wetter', 'signifikantes.wetter', 'luftdruck'};
    forecast_details = {'x', 'richtung', 'staerke', 'min', 'max', ...
                    'mittlere.temp.prog' ...
                    'dauer', 'einstrahlung', 'boeen', 'bodenfrost', ...
                    'gefrierender.regen', 'menge', 'kaelte', 'hitze', ...
                    'bodennebel', 'wahrscheinlichkeit', 'niederschlag'};
    forecast_interval = {'1', '2', '3', 'all'};

% Determine if the input values are member of those lists defined above. If
% this is the case val_inpt values will be 1.
    val_inpt(3) = ismember(fc_def{1}, forecast_scope);
    val_inpt(4) = ismember(fc_def{2}, forecast_details);
    val_inpt(5) = ismember(fc_def{3}, forecast_interval);
% If any input value doesn't exist in the list, create error message.
    if val_inpt(3) == 0
        err_msg{3} = ['Bitte ueberpruefen Sie den Prognosebereich: ' fc_def{1}];
    end
    if val_inpt(4) == 0
        err_msg{4} = ['Bitte ueberpruefen Sie das Prognosedetail: ' fc_def{2}];
    end
    if val_inpt(5) == 0
        err_msg{5} = ['Bitte ueberpruefen Sie das Prognoseintervall: ' fc_def{3}];
    end
end
end

```

An dieser Stelle wird die Wetterdatenanfrage analysiert. Kommt in dem Ausdruck nicht zweimal ein Querstrich vor, ist die Eingabe schon fehlerhaft. Wenn doch, werden die drei einzelnen Bestandteile mit Listen abgeglichen und bei entsprechender Existenz keine Fehlermeldung ausgegeben.

```

if ~isempty(varargin)
    try
        a=datenum(varargin{1});
    end
end

```

```

        val_inpt(6) = 1;
    catch
        err_msg{6} = ['Bitte ueberpruefen Sie das Startdatum des'...
                     'Beobachtungsintervalls: ' varargin{1}];
        val_inpt(6) = 0;
    end
    try
        a=datetime(varargin{2});
        val_inpt(7) = 1;
    catch
        err_msg{7} = ['Bitte ueberpruefen Sie das Enddatum des'... '
                     'Beobachtungsintervalls: ' varargin{2}];
        val_inpt(7) = 0;
    end

    if val_inpt(6) == 1 && val_inpt(7) == 1
% Calculate the difference of days between the observation start and end
% date.
        diff_days = days365(varargin{1},varargin{2})*24;

% If the difference is negative, the end date comes previous to the start
% date which is not possible.
        if diff_days < 0
            val_inpt(8) = 0;
            err_msg{8} = (['Das Startdatum fuer den Beobachtungszeitraum'...
                           'muss vor dem Enddatum liegen! Bitte'...
                           'korrigieren Sie die Datumseingabe.']);
% Same procedure with the start date, which never comes previous to the
% current date.
            elseif days365(date,varargin{1}) < 0
                val_inpt(8) = 0;
                err_msg{8} = ('Das Startdatum liegt in der Vergangenheit!');
            else
                val_inpt(8) = 1;
            end
        end
    end
end

```

Die Überprüfung des Observationszeitraums erfolgt dahingehen, dass der Ausdruck ein Datumsformat darstellen muss, den MATLAB mittels **datetime** konvertieren kann. Ist dies nicht möglich, liegt ein Fehler vor. Sind die Datumsformate korrekt, so kann es immer noch der Fall sein, dass das Startdatum in der Vergangenheit oder vor dem Enddatum liegt. Auch hier werden entsprechende Fehlermeldungen generiert.

```

% #### Check resolution and update intervall ####
resolution_values = {'6','1','0.5','0.25','0.08'};
updateinterval_values = {'6','12','24'};

val_inpt(9) = ismember(num2str(varargin{3}),resolution_values);

if val_inpt(9) == 0
    err_msg{9} = (['Fuer die Aufloesung koennen nur folgende Werte'...

```

```

                                'eingegeben werden: 6, 1, 0.5, 0.25, 0.08!']);
end

    val_inpt(10)    = ismember(num2str(varargin{4}),updateinterval_values);

    if val_inpt(10) == 0
        err_msg{10}    = (['Fuer das Updateintervall koennen nur folgende'...
                            'Werte eingegeben werden: 6, 12, 24!']);
    end
end

% If no error exists don't return a error msg or if there are less than 6
% error messages, make the last error message empty so it could be
% displayed in the fprintf command.
    if true(val_inpt)
        err_msg        = NaN;
    elseif size(err_msg,2) < 10
        err_msg{10}    = [];
    end
end

```

Im letzten Teil der Überprüfung werden die Werte der zeitlichen Auflösung und des Updateintervalls wieder mit Listen abgeglichen. Ist der Gültigkeitsvektor in der boolschen Überprüfung wahr, werden keine Fehlermeldungen ausgegeben. Andernfalls wenn weniger als 10 Fehlermeldungen angefallen sind, muss der letzte Vektoreintrag der Fehlermeldungen leer sein. Dies ist erforderlich, um den nachfolgenden Print-Befehl ausführen zu können.

3.2.4 Verfügbarkeitsprüfung der seriellen Schnittstelle

```

% Check whether a variable serial interface already exists in the base
% workspace. If true delete this variable and all other available serial
% interfaces to be sure to build up the neccessary serial interface.
if evalin('base', ('exist(''serial.interface'')')) == 1
    evalin('base', ('delete(''serial.interface'')'));
    evalin('base', 'clear serial.interface');
    evalin('base', 'delete(instrfind)');
end

% Check for available COM Ports, if COM6 is not available print message
av_com_ports    = instrhwinfo('serial');
com_port_av     = find(ismember(av_com_ports.AvailableSerialPorts,'COM6'));

if isempty(com_port_av)
    fprintf(2,'COM6 Port ist nicht verfuegbar!\n', char(10));
    return;
end

```

Existiert bereits eine Variable *serial_interface* im Base-Workspace, so wird diese gelöscht. Zudem werden alle anderen seriellen Schnittstellen gelöscht um zu vermeiden, dass bereits eine andere

serielle Schnittstelle den COM6 Port belegt hat. Danach wird die Verfügbarkeit des COM6 Ports festgestellt. Ist dies nicht der Fall, wird eine Fehlermeldung an den Nutzer ausgegeben und die Funktion beendet.

3.3 Aufbau der seriellen Schnittstelle

```
% Open serial interface
open_serial_port( 'COM6', 19200, 8, 'even', 1 );
```

Die Funktion `open_serial_port` enthält bereits alle in der Tabelle 1.2 auf Seite 13 festgelegten Schnittstellenparameter.

```
function [ ] = open_serial_port( com_address, baudrate, databits, parity, stopbit )
%This function establishes the serial interface for the modbus
%communication channel.
%   For the HWK Kompakt COM address has to be COM6, Baudrate = 19200,
%   Databits = 8, Parity = 'even' and Stopbit = 1

% Creates the serial interface
serial_interface = serial(com_address, 'BaudRate',baudrate, 'DataBits', ...
                           databits, 'Parity',parity, 'StopBits',stopbit);

% Export variable to base workspace
assignin('base', 'serial_interface', serial_interface);

% Open the serial interface
fopen(serial_interface);

fprintf(['Serielle Schnittstelle wurde eingerichtet unter der Variable\n'...
```

MATLAB bietet für den Aufbau einer seriellen Schnittstelle eine Funktion namens `serial` an. Diese wird hier zur Erstellung der Variable `serial_interface` angewandt. Die Variable wird dem Base-Workspace zugewiesen und die serielle Schnittstelle mit dem Befehl `fopen` geöffnet. Der Nutzer wird über den Schnittstellenaufbau informiert.

3.4 Abgleich der Wetterregion im Register

```
city_id_reg = read_com_set(device_id, {'city_id'});

% If no value is detected for the city id register, the required city id
% will be written to that register. If the existent register value doesn't
% match the required value it will be overwritten.
if isempty(city_id_reg)
    fprintf('Es befindet sich kein Wert in Register 112!\n');
```

```

write_com_set( device_id, city_id, {'city_id'} );
fprintf(['Neue CityID %u wurde in das Register geschrieben.\n'...
        'Es wird ein paar Stunden dauern, bis alle Register aktualisiert wurden.\n\n'],...
        city_id);
elseif city_id ~= city_id_reg
    prompt = ['Die vorhandene City ID entspricht nicht der in der Funktion'...
             'uebergebenen ID.\n Moechten Sie fortfahren? Y/N [Y]: '];
    str = input(prompt, 's');
    if isempty(str)
        str = 'Y';
    end
    if strcmp(str, 'Y') == 1
        write_com_set( device_id, city_id, {'city_id'} );
        fprintf(['Neue CityID %u wurde in das Register geschrieben.\n Es wird'...
                'ein paar Stunden dauern, bis alle Register aktualisiert wurden.\n\n'], city_id);
    else
        fprintf(2, 'Der Funktionsaufruf wurde abgebrochen.\n', char(10));
        return;
    end
end
end

```

Da der Wechsel einer Wetterregion unter Umständen bis zu drei Tagen dauern kann, bis alle anliegenden Werte Gültigkeit besitzen, wird in diesem Teil des Codes zuerst die bereits eingetragene Stadt-ID ausgelesen und mit der in dem Funktionsaufruf Angegebenen verglichen. Weichen die beiden Werte voneinander ab, so wird der Nutzer per Tastatureingabe aufgefordert dem Wetterregionenwechsel zuzustimmen. Er hat somit die Gelegenheit einen versehentlichen Wechsel abzubrechen. Anschließend wird der neue Wert im Register eingetragen.

3.5 Festlegung der Timerparameter und Starten des Timers

```

if ~isempty(varargin)

% Split the datestring into single elements
    start_observation      = regexp(start_observation, '-', 'split');
    end_observation        = regexp(end_observation, '-', 'split');

% Calculate day difference in hours between the observation start date and
% end date. Further determine start and end date of current day.

    diff_days              = days365(start_observation, end_observation)*24;
    end_of_day              = datevec(date)+[0 0 0 24 0 0];
    start_of_day            = datevec(now);

% When the observation start date equals current date, calculate the
% remaining hours from calling the function to the end of current day. The
% number of update cycles results from the sum of remaining hours from the
% current day and hours between the days after current day to end of
% observation divided by the update interval. You have to add 1 for the

```



```

% first request executed immediatly with this function call.
% If the observation start date is in the future, and observation start
% date and end date are equal, 24 hours are available. Update cycle number
% is the result of the division diffdays/update_interval. Start delay is
% calculated from the sum of remaining hours of current date and difference
% of days in hours till start of observation.

if strcmp(start_observation,date) == 1
    diff_today          = etime(end_of_day,start_of_day)/3600;
    update_cycle_number = floor((diff_today+diff_days)/update_interval)+1;
    assignin('base','update_cycle_number',update_cycle_number);
else
    if datenum(start_observation) == datenum(end_observation)
        diff_days      = 24;
    end
    diff_today          = etime(end_of_day,start_of_day);
    diff_days2start     = days365(date,start_observation);
    start_delay         = uint32(diff_today+diff_days2start*86400);
    update_cycle_number = floor(diff_days/update_interval);
    assignin('base','update_cycle_number',update_cycle_number);
end

```

Um die Eigenschaft des wiederholten Datenabrufs zu implementieren wurde das Timer-Objekt von MATLAB implementiert. Es bietet den Vorteil im Hintergrund zu laufen ohne dabei MATLAB komplett zu blockieren. Für die Initialisierung dieses Objekts müssen zuerst ein paar Parameter ermittelt werden. Insbesondere betrifft dies das Timerintervall, die Anzahl der Timerausführungen und die Timerverzögerung. Das Timerintervall lässt sich einfach dadurch berechnen indem die Variable *update_interval* mit der Anzahl an Sekunden für eine Stunde multipliziert wird. Die Anzahl der Timerausführungen ergibt sich dann durch die zur Verfügung stehenden Stunden im Beobachtungszeitraum dividiert durch die Länge des Timerintervalls. Liegt der Beobachtungszeitraum in der Zukunft, so muss die Timerverzögerung genau die Länge vom Funktionsaufruf bis zum Startdatum aufweisen. Bei der Berechnung dieser Angaben sind die MATLAB Funktionen `days365`, `datevec` sowie `etime` äußerst nützlich.

```

% Requests start with a 3 sec delay. The function to be executed after
% the waiting period is send_loop, which triggers the communication
% between Matlab and the weather station. The stop function deletes the
% timer object after all tasks have been executed.
t = timer;

if strcmp(start_observation,date) == 1
    t.StartDelay = 3;
else
    t.StartDelay = start_delay;
end
t.TimerFcn = {@send_loop, size_table_data, fc_def, device_id, filepath, city_name};
t.StopFcn = {@stop_timer, filepath, city_name, resolution};

```

Mit der Zuweisung „ $t = \text{timer}$ “ wird in der Variablen t das Timer-Objekt erzeugt. Die Eigenschaften können dann ähnlich einer Struktur in MATLAB aufgerufen und bestimmt werden. Zu bestimmen sind die Timerverzögerung ($t.StartDelay$), das Timerintervall ($t.Period$), die Timerausführungen ($t.TasksToExecute$), sowie die Ausführungsmethode ($t.ExecutionMode$). Die Ausführungsmethode „fixedRate“ garantiert, dass in genau gleichen Timerintervallen die in der Timerfunktion ($t.TimerFcn$) definierte Funktion ausgeführt wird. Wird der Timer durch einen Fehler oder durch einen Abbruchbefehl unterbrochen, so legt man in der Timerstopfunktion fest, was geschehen soll.

```
function [ ] = stop_timer(mTimer,~, filepath, city_name, resolution)
%Deletes timer object, serial interface and saves weather_data container to specified folder
% Detailed explanation goes here

fprintf(['Automatischer Abruf fuer den angegebenen Beobachtungszeitraum\n' ...
        'wurde beendet.\n']);

delete(mTimer)

% Define filename as city_name-weather_data-current_date-current_unix-time
% and save to specified filepath
filename = strcat(filepath, '\', city_name, '-', strrep(num2str(resolution), ...
    '.', '_'), '_weather_data-', date, '-', num2str(date2utc(datevec(now))), '.mat');
weather_data = evalin('base', 'weather_data');
save(filename, 'weather_data', '-mat');

close_serial_port();
evalin('base', 'clear update_cycle_number');
end
```

Kommt es zu einem Timerabbruch, so wird eine Meldung an den Nutzer ausgegeben, das Timer-Objekt gelöscht und der Datencontainer mit den fortlaufenden Werten abgespeichert. Außerdem wird die serielle Schnittstelle beendet. Der Dateiname für den Datencontainer setzt sich zusammen aus der Wetterregion, der angewandten zeitlichen Auflösung, dem aktuellen Datum und dem aktuellen Datum in Unix Zeitformat.

3.6 Abschieken der MODBUS-Anfragen

```
function[ ] = send_loop(obj, event, t, fc_def, dev_id, f_path, city, u_c_n, res,...
    lng, lat )
```

Die Funktion, die das sequentielle Abschieken und Verarbeiten der einzelnen MODBUS Nachrichten übernimmt wird hier erläutert.

```
h = waitbar(0, 'Please wait while receiving data...');
```

```

daychange_flag = evalin('base','daychange_flag');
daychange_counter = evalin('base','daychange_counter');
w_dat = evalin('base','weather_data');

% For loop to process every forecast definition(fc_def).
for r = 1:t

    fc_int          = regexp(fc_def{r}, '-', 'split');
% For the first loop determine if the datacontainer weather_data(w_dat) has
% stored previous data by analyzing the last stored recording time stamp.
% If there is such data, compare timestamp with current date. If it is not
% equal, increase daychange_counter and set daychange_flag true. Make both
% variables available in base workspace.

    if r == 1
        if ~isempty(w_dat.(fc_int{1}).(fc_int{2}).unix_t_rec)
            t_rec = w_dat.(fc_int{1}).(fc_int{2}).unix_t_rec(...
                size(w_dat.(fc_int{1}).(fc_int{2}).unix_t_rec,2));

            if days365(utc2date(t_rec),date) ~= 0
                daychange_flag = 1;
                daychange_counter = daychange_counter + 1;
            else
                daychange_flag = 0;
            end
            assignin('base','daychange_flag',daychange_flag);
            assignin('base','daychange_counter',daychange_counter);
        end
    end
end

```

Nachdem die Eingabeparameter übergeben wurden, werden die im Base-Workspace vorhandenen Variablen *daychange_flag*, *daychange_counter* sowie der Datencontainer *w_dat* für die Langzeitar- chivierung in dieser Funktion bereitgestellt. Eine for-Schleife durchläuft dann alle Wetterdaten- anfragen die in *fc_def* definiert wurden. Beim ersten Durchlauf muss geprüft werden, ob bereits zu einem früheren Zeitpunkt Daten ausgelesen wurden damit ein Tageswechsel signalisiert wer- den kann. Hierzu wird der letzte Zeitstempel des letzten Datenabrufs, sofern vorhanden, mit dem aktuellen Datum verglichen. Ist das Ergebnis ungleich Null, liegt ein Tageswechsel vor und die Variablen werden entsprechend angepasst und im Base-Workspace aktualisiert.

```

forecast_days          = fc_int{1,3};

if strcmp(fc_int{1,2}, 'mittlere_temp_prog') == 1
    switch forecast_days
        case '1'
            start_reg    = {'heute' 'am0_00'};
            end_reg      = {'heute' 'pm11_00'};
        case '2'
            start_reg    = {'heute' 'am0_00'};

```

```

        end_reg      = {'erster_folgetag' 'pm11_00'};
    case '3'
        start_reg     = {'heute' 'am0_00'};
        end_reg       = {'zweiter_folgetag' 'pm11_00'};
    case 'all'
        start_reg     = {'heute' 'am0_00'};
        end_reg       = {'dritter_folgetag' 'pm11_00'};
    end
elseif strcmp(fc_int{1,1}, 'solarleistung') == 1 || ...
    strcmp(fc_int{1,1}, 'luftdruck') == 1
if str2double(forecast_days) > 1
    warning(['Fuer die Solarleistungs- und Luftdruckprognose' ...
        ' werden nur Werte fuer den heutigen Tag und den' ...
        ' ersten Folgetag bereitgestellt.'])
    forecast_days = 'all';
end
switch forecast_days
    case '1'
        start_reg     = {'heute' 'morgen'};
        end_reg       = {'heute' 'abend'};
    case 'all'
        start_reg     = {'heute' 'morgen'};
        end_reg       = {'erster_folgetag' 'abend'};
    end
else
    switch forecast_days
        case '1'
            start_reg     = {'heute' 'morgen'};
            end_reg       = {'heute' 'abend'};
        case '2'
            start_reg     = {'heute' 'morgen'};
            end_reg       = {'erster_folgetag' 'abend'};
        case '3'
            start_reg     = {'heute' 'morgen'};
            end_reg       = {'zweiter_folgetag' 'abend'};
        case 'all'
            start_reg     = {'heute' 'morgen'};
            end_reg       = {'dritter_folgetag' 'abend'};
        end
    end
end

% if size(fc_int,2) < 5
%     end_reg      = start_reg;
% else
%     end_reg      = fc_int(1,5:6);
% end

fc_int      = {fc_int{1,1:2}, start_reg{:}, end_reg{:}};

```

Im nächsten Schritt wird das Intervall für die auszulesenden Wetterdaten zusammengestellt. Hierbei muss unterschieden werden zwischen den stündlichen Werten der mittleren Lufttempe-

ratur, dem begrenzten Prognosehorizont von Luftdruck und Solarleistung und den verbleibenden Daten. Gibt der Nutzer eine zu hohe Zahl für Luftdruck oder Solarleistung ein, so wird auf die möglichen Werte angepasst und eine Warnung ausgegeben.

```
% Determine register addresses and number of registers to be processed
start_reg_address = get_reg_address( fc_int{1}, fc_int{2}, start_reg );
end_reg_address   = get_reg_address( fc_int{1}, fc_int{2}, end_reg );

quantity_reg_addresses = reg_num(start_reg_address, end_reg_address);
% Generate modbus message
modbus_pdu = gen_msg( dev_id, start_reg_address, ...
                    quantity_reg_addresses, 'rsr' );
% Read the connection quality
con_qual = read_com_set('03',{'quality'});
% Write message on interface and read and process response
txdata = send_and_receive_data(modbus_pdu, fc_int, ...
                               res, con_qual, lng, lat);

waitbar(r/t,h)

end
```

In diesem Abschnitt wird die MODBUS Nachricht erstellt und einer Funktion übergeben, die das Schreiben und Lesen auf der seriellen Schnittstelle übernimmt. Die Startadresse des Registers ergibt sich aus dem Prognosebereich, dem Prognosedetail und dem Anfangszeitpunkt zu dem die Werte ausgelesen werden sollen. Die Endadresse wird im gleichen Verfahren ermittelt. Die Differenz aus den beiden Adressdaten ergibt die Registeranzahl. Gemäß der MODBUS Spezifikation liegen nun alle Daten vor, die zur Kommunikation notwendig sind. Lediglich der Cyclic Redundancy Check fehlt noch. Dieser wird in der Funktion **gen_msg** erstellt und der PDU zusammen mit der Slave-ID angeheftet. Wie die momentane Verbindungsqualität zum Zeitpunkt der Anfrage ist, wird kurz darauf abgerufen. Alle für die weitere Verarbeitung erforderlichen Daten werden nun an die Funktion **send_and_receive_data** übergeben.

```
% Save requested data in a separate file
new_data = evalin('base','new_data');
filename = strcat(f_path,'\ ',city,'-',strrep(num2str(res),'.','_'),'_new_data-',...
                date,'_',num2str(date2utc(datevec(now))),'.mat');
save(filename,'new_data','-mat');
% Reset new_data container
new_data = [];
for z = 1:t
    new_data = create_data_struct(fc_def{z}, new_data, 'new_data');
end
assignin('base','new_data',new_data);
% Update the remaining number of requests
if ~isempty(u_c_n)
    u_c_n = evalin('base','u_c_n');
```

```

        u_c_n = u_c_n-1;
        fprintf('Noch %u ausstehende Abfrage(n).\n',u_c_n)
        assignin('base','u_c_n',u_c_n);
    end
    close(h);
end

```

Ist die for-Schleife durchlaufen und alle Daten liegen in den Datencontainern, wird der vollzogene Abruf in dem Container *new_data* mit entsprechendem Dateinamen abgespeichert. Der Dateiname setzt sich dabei aus dem Pfad zum Speicherort, dem Stadtnamen, der gewünschten Auflösung, dem aktuellen Datum und einem Datum in Unix Zeitformat zusammen. Ist das Speichern abgeschlossen, wird der Datencontainer geleert und neu initialisiert. Der letzte Schritt in diesem Programmteil ist das Update der noch verbleibenden Anzahl an Datenabrufen, welche in der Variable *u_c_n* (update cycle number) hinterlegt sind.

3.7 Senden und Empfangen

```

function [ value ] = send_and_receive_data( modbus_msg, field_name, res,...
                                            con_qual, lng, lat )
%Writes and reads modbus message on serial interface
%   Detailed explanation goes here

% Makes serial interfaces available in local workspace
serial_interface = evalin('base','serial_interface');

% Formats the modbus message into serial interface readable structure
txdata = format_modbus_msg(modbus_msg);

% Write message
fwrite(serial_interface,txdata);

pause(1);

% Check how many bytes have been received on serial interface
if serial_interface.BytesAvailable == 0
    bytes_num = 8;
else
    bytes_num = serial_interface.BytesAvailable;
end

% Read message data as unsigned values
[rxdata] = fread(serial_interface, bytes_num, 'uint8');

% Call rxdata processing
[value, error_msg] = rxdata_processing( rxdata, modbus_msg, field_name, res,...
                                         con_qual, lng, lat );

end

```

In dem Funktionsworkspace der Funktion `send_and_receive_data` muss die serielle Schnittstelle zugänglich sein. Um die in der Funktion `send_loop` generierte Nachricht über die Schnittstelle abschicken zu können, muss sie auf ein geeignetes Format gebracht werden. Dann erst kann die Nachricht mit dem Befehl *fwrite* übermittelt werden. Es wird 1 Sekunde auf die Antwort gewartet. Das bestimmen der empfangenen Bytes und die Angabe beim Lesebefehl `fread` beschleunigt die Kommunikation enorm. Jetzt liegt die Antwortnachricht des Servers in der Variablen *rx_data* vor. Die darin enthaltenen Werte wurden als unsigned integer 8 bit empfangen.

```
function [ txdata ] = format_modbus_msg( modbus_msg_crc )
%Format modbus pdu to 'fwrite' readable structure
% Detailed explanation goes here
for e = 2:2:(size(modbus_msg_crc,2)-2)
    if e == 2
        temp1 = modbus_msg_crc(1,1:e);
        temp2 = modbus_msg_crc(1,e+1:end);
        string = strcat(temp1,':',temp2);
    else
        temp2 = modbus_msg_crc(1,e+1:end);
        string = strcat(string(1,1:(size(string,2)-size(temp2,2))),':',temp2);
    end
end
txdata = regexp(string,':','split');
txdata = hex2dec(txdata);
end
```

Die Funktion `format_modbus_msg` transformiert den String der MODBUS-Nachricht in einen 8-zeiligen Vektor mit dezimalen Werten die den hexadezimalen Werten des Strings entsprechen.

3.8 Rx-Datenverarbeitung

3.9 Datenverarbeitung

Prognosebereich	Update-zeitpunkt	Prognoseintervalle			
		00:00:00-05:59:59	06:00:00-11:59:59	12:00:00-17:59:59	18:00:00-23:59:59
Temperatur Min. in °C	17.01.2014 03:26 Uhr	1	3	3	1
	17.01.2014 09:26 Uhr	2	3	3	2
	17.01.2014 15:26 Uhr	3	3	3	2
	17.01.2014 21:26 Uhr	3	3	2	1
	18.01.2014 03:26 Uhr	1	1	4	1
	18.01.2014 09:26 Uhr	-1	0	4	1
	18.01.2014 15:26 Uhr	-2	-2	3	1
	18.01.2014 21:26 Uhr	-2	-2	1	0
Luftdruck in hPa	17.01.2014 03:26 Uhr	1007	1005	1005	1006
	17.01.2014 09:26 Uhr	1007	1006	1004	1005
	17.01.2014 15:26 Uhr	1007	1006	1005	1006
	17.01.2014 21:26 Uhr	1007	1006	1005	1007
	18.01.2014 03:26 Uhr	1007	1004	1001	1001
	18.01.2014 09:26 Uhr	1007	1004	1000	1000
	18.01.2014 15:26 Uhr	1007	1004	1001	1001
	18.01.2014 21:26 Uhr	1007	1004	1001	1001

Tabelle 3.1: Datenverlust bei kleinem Updateintervall am Beispiel minimaler Temperatur und Luftdruck

3.10 Funktionsaufbau

Kapitel 4

Datenanalyse

Teil III

Zusammenfassung und Ausblick

Anhang

Anhang A

Aufbau der Wetterstation

Prognose-Bereich	Prognosedetail	Einheit	Wertebereich	Auflösung	Prognoseintervall	zeitliche Auflösung	Registeradresse	Info
Temperatur	Min. Lufttemperatur	°C	< -60 bis > 65	1	3 Tage	6h	420 - 435	Mittlerer Wert 2m über Erdboden
Temperatur	Max. Lufttemperatur	°C	< -60 bis > 65	1	3 Tage	6h	400 - 415	Mittlerer Wert 2m über Erdboden
Temperatur	Mittlere Lufttemperatur	°C	< -60 bis > 65	1	3 Tage	1h	000 - 095	Mittlerer Wert 2m über Erdboden
Temperatur	Lokale Lufttemperatur	°C	< -60 bis > 65	1	Aktuell	1s	097	Standortabhängig
Niederschlag	Menge	l/m ²	0-60	dyn.	3 Tage	6h	140 - 155	Mittlerer Wert
Niederschlag	Wahrscheinlichkeit	%	0 - 100	10	3 Tage	6h	160 - 175	Mittlerer Wert
Solarprognose	Sonnenscheindauer	h	0 - 6	1	1 Tag	6h	180 - 187	Mittlerer Wert Globalstrahlung
Solarprognose	Solare Einstrahlung	W/m ²	0 - 1200 / > 1200	25	1 Tag	6h	190 - 197	Mittlerer Wert bezogen auf NN
Luftdruck	Min. Lufttemperatur	hPa	< 938 - > 1063	1	1 Tag	6h	240 - 247	Mittlerer Wert 10m über Erdboden
Windprognose	Stärke	Bft	0 - 12	1	3 Tage	6h	200 - 215	N/NO/O/SO/S/SW/W/NW
Windprognose	Richtung		1 - 8	1	3 Tage	6h	220 - 235	0=keine Böen, 1=45km/h, 2=72km/h, 3=99km/h
Markantes Wetter	Böen		0,1,2,3	1	3 Tage	6h	310 - 325	0=Wahrscheinlichkeit<=50%, 1=>50%
Markantes Wetter	Bodennebel		0,1	1	3 Tage	6h	250 - 265	0=Wahrscheinlichkeit<=50%, 1=>50%
Markantes Wetter	Gefrierender Regen		0,1	1	3 Tage	6h	270 - 285	0=Wahrscheinlichkeit<=50%, 1=>50%
Markantes Wetter	Hitze	°C	0,1,2,3,4	1	3 Tage	6h	350 - 365	0=<27°C, 1=27-31°C, 2=32-40°C, 3=41-53°C, 4=>54°C
Markantes Wetter	Kälte	°C	0,1,2,3,4	1	3 Tage	6h	370 - 385	0=keine Info, 1=<-15°C, 2=<-20°C, 3=<-25°C, 4=<-30°C
Markantes Wetter	Bodenfrost		0,1	1	3 Tage	6h	290 - 305	0=Wahrscheinlichkeit<=50%, 1=>50%
Markantes Wetter	Niederschlag		0,1,2,3	1	3 Tage	6h	330 - 345	0=keine Info, 1=10mm, 2=50mm, 3=keine Info
Signifikantes Wetter			1 - 15	1	3 Tage	6h	120 - 135	1=sonnig/klar, 2=leicht bewölkt, 3=vorwiegend bewölkt, 4=bedeckt, 5=Wärmegewitter, 6=starker Regen, 7=Schneefall, 8=Nebel, 9=Schneeregen, 10=Regenschauer, 11=leichter Regen, 12=Schneeschauer, 13=Frontengewitter, 14=Hochnebel, 15=Schneeregensdauer

Tabelle A.1: Detaillierte Datenstruktur der Wetterstation[1, S. 17-26]

Anhang B

Das MODBUS Protokoll

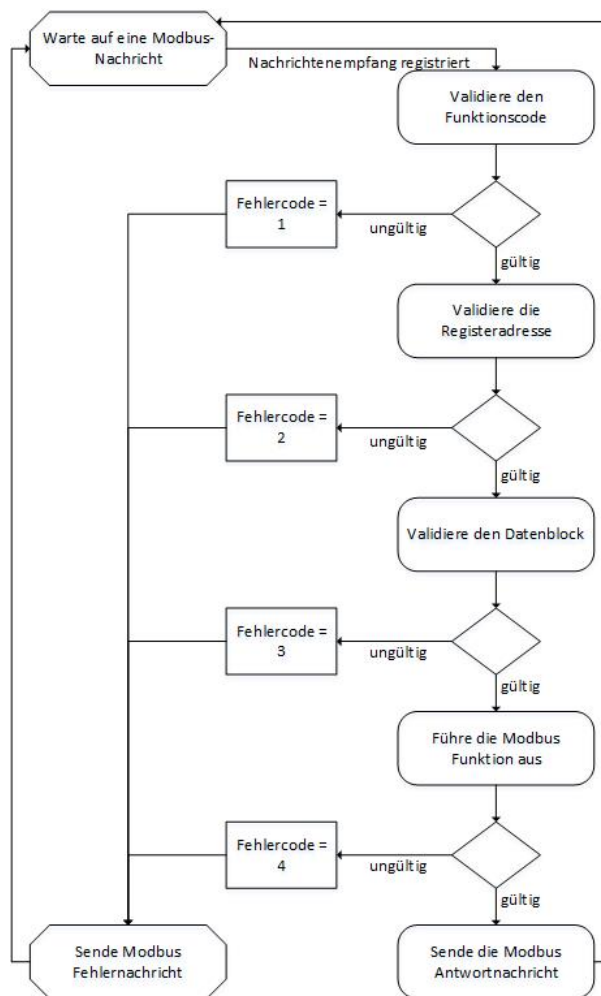


Abbildung B.1: Ablaufdiagramm für die MODBUS Nachrichtenüberprüfung

				Function Codes		
				Code	Sub Code	Hex
Data Access	Bit Access	Physical <u>Discret</u> Inputs	Read Discrete Inputs	02		02
		Internal Bits or Physical coils	Read Coils	01		01
			Write Single Coil	05		05
			Write Multiple Coil	15		0F
	16 bit access	Physical Input Registers	Read Input Register	04		04
		Internal Registers or Physical Output Registers	Read Holding Registers	03		03
			Write Single Registers	06		06
			Write Multiple Registers	16		10
			Read/Write Multiple Registers	23		17
			Mask Write Register	22		16
			Read FIFO queue	24		18
			File record access	Read File Record	20	
	Write File Record	21			15	
	Diagnostics		Read Exception status	07		07
			Diagnostic	08	00-18,20	08
			Get Com Event Counter	11		
Get Com Event Log			12		0C	
Report Server ID			17		11	
Read Device Identification			43	14	2B	
Other		Encapsulated Interface Transport	43	13,14	2B	
		<u>CANopen</u> General Reference	43	13	2B	

Abbildung B.2: Übersicht der zur Verfügung stehenden Funktionscodes in MODBUS

Anhang C

Abkürzungsverzeichnis

NN Normal Null Meeresspiegel

hPa Hektopascal

Bft Beaufort

mm Millimeter

km Kilometer

h Stunde

C Celsius

N Nord

NO Nordost

O Osten

SO Suedost

S Sueden

SW Suedwest

W Westen

NW Nordwest

Literaturverzeichnis

- [1] HKW-Elektronik GmbH. *Wetterprognose-Station Kompakt WS-K xx Modbus*. HKW-Elektronik GmbH, Industriestraße 12, D-99846 Seebach/Thur, stand 23.11.2012 edition, November 2012.
- [2] Nationale Plattform Elektromobilität (NPE). *Fortschrittsbericht der Nationalen Plattform Elektromobilität*, page 3. Number 3. Gemeinsame Geschäftsstelle Elektromobilität der Bundesregierung (GGEMO), Scharnhorststraße 34–37, 10115 Berlin, 2012.
- [3] Bayern Innovativ Gesellschaft für Innovation und Wissenstransfer mbH. <http://www.elektromobilitaet-verbindet.de/>, 1 2014.
- [4] Modbus Organization, Inc. *MODBUS APPLICATION PROTOCOL SPECIFICATION*. Modbus Organization, Inc., PO Box 628 Hopkinton, MA 01748, v1.1b3 edition, April 2012.
- [5] Gerd Küveler and Dietrich Schwoch. *Informatik für Ingenieure und Naturwissenschaftler*. Viewegs Fachbücher der Technik. Vieweg, Braunschweig [u.a.], 5 edition, 2007.