TagTrans Programmierhandbuch

Rev 3.0

Inhaltsverzeichnis

TagTrans V2.1 FW 0.12

1 Power Management

Abb. TagTrans Teilebeschreibung

Sobald der Haupt-Teil mit dem Akku-Teil (siehe Abb. X) verbunden ist, wird der TagTrans fortlaufend versorgt, wobei ab diesem Zeitpunkt mehrere Betriebsmodi möglich sind. In diesem Kapitel werden diese einzeln beschrieben.

1.1 Verbindungsaufbau

Während diesem Modus, versucht standardmäßig der TagTrans eine Bluetooth -Verbindung mit einem Host aufzubauen. Über drei Möglichkeiten gelangt man in diesen Modus:

1. Der Akku-Teil wird an den Haupt-Teil angeschlossen – **Reset**.
2. Ausgehend vom **Schlafmodus**:

Man führt die „manuelle Weck-Bewegung“ für mindestens 2 Sekunden durch – **Reset**.

3. Ausgehend vom **Vollbetrieb** oder **Bereitschaftsmodus**:

Host - seitige Verbindungstrennung.

1.1.1 Anzeige/Signale

Im Normalfall, wenn der Akku voll geladen ist, leuchtet eine LED **grün** – diese LED ist die **Batteriestands-Anzeige**. Wenn diese LED **orange** leuchtet, ist der Akku-Stand mittelmäßig bis niedrig und wenn sie einige Sekunden **rot blinkt** muss der Akku geladen werden.

Nach einem Reset ertönt zusätzlich ein Signal als Hinweis für eine erfolgreiche Initialisierung des TagTrans.

1.1.2 Dauer

Ein Zeitzähler **PwrTmr** wird standardmäßig – nach einem Reset - mit einem Wert von 180s (set **t1**) geladen. Solange besteht die Möglichkeit mit dem TagTrans eine Verbindung aufzubauen. Ist der Zeitzähler auf Null, wechselt der TagTrans in den **Schlafmodus**. Falls jedoch eine Verbindung aufgebaute werden konnte, wechselt das Gerät in den **Vollbetrieb**.

Mit einer kurzen „manuellen Weck-Bewegung“ wird der **PwrTmr** mit 60s (set **t2**) geladen.

1.1.3 Verbrauch

Während dem Verbindungsaufbau wird lediglich das **BT-Modul** versorgt. Das HF-Modul bleibt inaktiv – ca. **50 .. 150 mA** (entspricht etwa 50 Stunden).

1.2 Vollbetrieb

Im Vollbetrieb sind alle Module aktiv. Die Kommunikation zwischen HF-Modul und Host ist ab jetzt aufrecht. Jedes gesendete Zeichen (Byte) vom Host wird nun direkt an das HF-Modul beziehungsweise vice versa über TagTrans weitergeleitet. Auch die sog. „%-Befehle“ werden von TagTrans akzeptiert – siehe Kap. „**TagTrans Befehle**“. Über zwei Möglichkeiten gelangt man in den Vollbetrieb:

1. Vom **Verbindungsaufbau** ausgehend:
   Host-seitige Verbindung herstellen.
2. Vom **Bereitschaftsmodus** ausgehend:
   Kurze „manuelle Weck-Bewegung“ oder senden eines Zeichens an TagTrans.

1.2.1 Anzeige/Signale

Die **Batteriestands-Anzeige** leuchtet entweder **grün oder orange**. Eine zweite LED – die **Verbindungs-Anzeige**, leuchtet in diesem Modus **blau** und signalisiert dem Benutzer, dass eine Verbindung mit einem Host besteht. Falls eine **Übertragung** stattfindet addiert sich kurzzeitig die Farbe rot hinzu, also **violett**.

Ein hohes **1-Sekunden Signal** ertönt, falls die Verbindung zuvor aufgebaut werden musste.

1.2.2 Dauer

Nach einem Verbindungsaufbau wird ein zusätzlicher Zeitzähler **RdTmr** mit 20s (set **t3**) geladen. Findet nun eine Kommunikation zwischen Host und HF-Modul statt oder führt man eine kurze „manuelle Weck-Bewegung“ durch, wird **RdTmr** wieder auf 20s (**t3)** und **PwrTmr** auf 60s (**t2)** zurückgesetzt.

Läuft der Zeitzähler **RdTmr** ab, wechselt TagTrans in den **Bereitschaftsmodus**. **RdTmr** muss immer kleiner als **PwrTmr** sein.

1.2.3 Verbrauch

Im **Vollbetrieb** wird am meisten verbraucht, da alle Module aktiv sind – ca. **200 .. 250 mA** (entspricht etwa 22 Stunden).

1.3 Bereitschaftsmodus

In diesem Modus steht die Verbindung zum Host, jedoch ist das HF-Modul deaktiviert. Das besondere an diesem Modus ist, dass man **schnell in den Vollbetrieb** wechseln kann, und dass der TagTrans **weniger Leistung** verbraucht als während dem Verbindungsaufbau. Es gibt nur eine Möglichkeit in diesen Modus zu gelangen:

1. Vom **Vollbetrieb** ausgehend: Der Zeitzähler **RdTmr** läuft ab.

1.3.1 Anzeige/Signale

Die **Batteriestands-Anzeige** leuchtet **grün oder orange** und die **Verbindungs-Anzeige** blinkt **blau**.

1.3.2 Dauer

Läuft der Zeitzähler **PwrTmr** ab, wechselt der TagTrans in den Schlafmodus und die Verbindung zum Host wird abgebrochen.

Eine Host-seitige Kommunikation oder eine kurze „manuelle Weck-Bewegung“ wechselt den TagTrans wieder in den Vollbetrieb.

1.3.3 Verbrauch

Durch das Deaktivieren des HF-Moduls und durch die bereits aufrechte Verbindung zwischen Host und TagTrans ist der Verbrauch zu allen anderen aktiven Modi am geringsten – ca. **40 .. 100mA** (entspricht etwa 72 Stunden).

1.4 Schlafmodus

Im Schlafmodus werden sämtliche Verbraucher abgeschaltet. Es gibt zwei Möglichkeiten in diesen Modus zu gelangen:

1. Aus dem **Vollbetrieb** ausgehend:

Senden des TagTrans-Befehls „%off“ (siehe Kap. „TagTrans-Befehle)

2. Aus dem **Bereitschaftsbetrieb** ausgehend:

Der Zeitzähler **PwrTmr** läuft ab.

1.4.1 Anzeige/Signale

Sowohl die Batteriestands-Anzeige als auch die Verbindungs-Anzeige leuchten nicht auf.

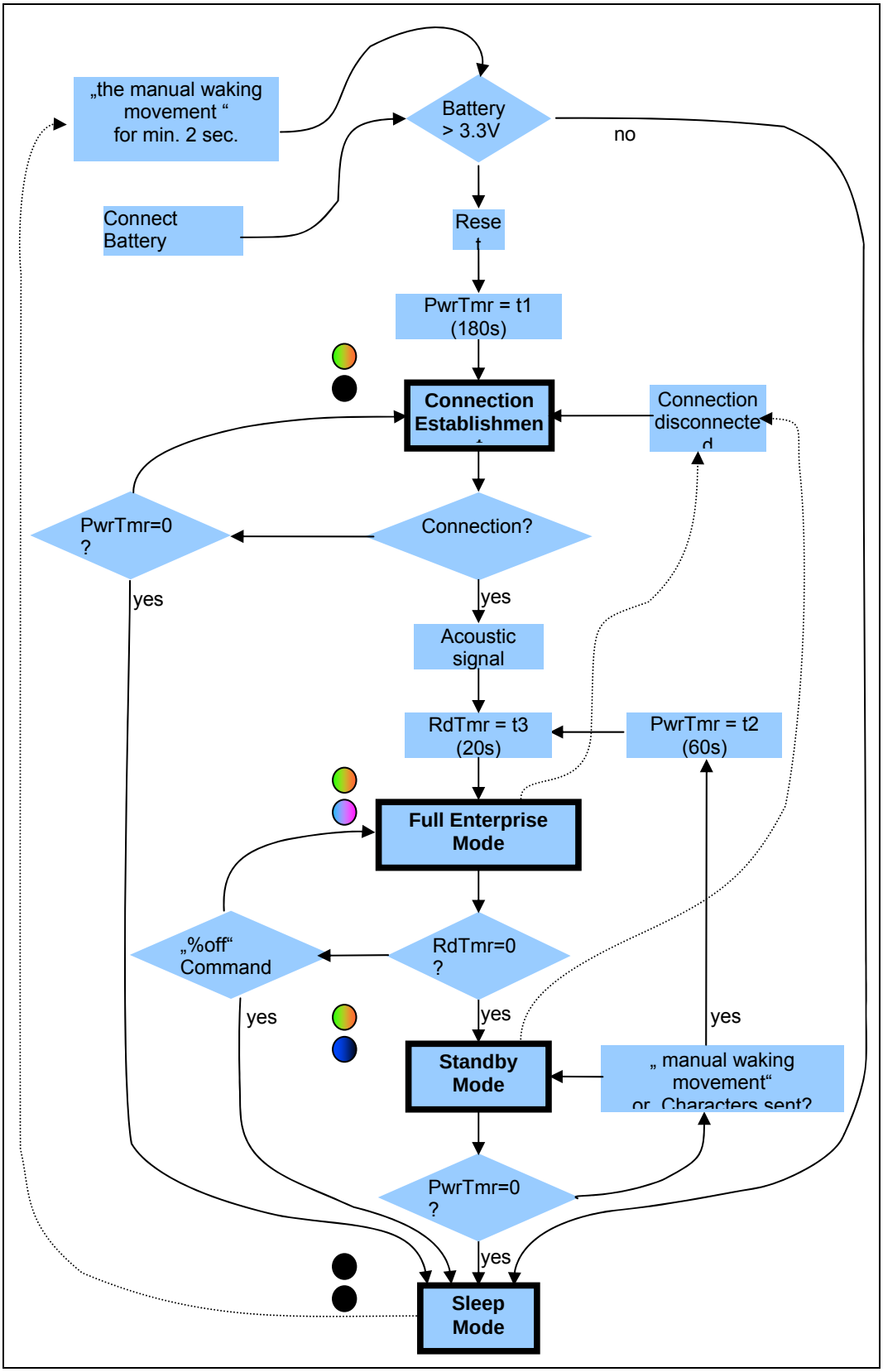
1.4.2 Dauer

Solange keine lange „manuelle Weck-Bewegung“ (ca. 2 sekundenlang) durchgeführt wird, bleibt der TagTrans im Schlafmodus.

1.4.3 Verbrauch

In diesem Modus gibt es praktisch keinen konstanten Stromfluss, das heißt dass TagTrans in diesem Modus praktisch nichts verbraucht - auch wenn der Akku-Teil angeschlossen ist. Durch kleinere mechanische Stöße kommt es zu kleineren Aufladeströmen.

1.5 Flussdiagramm


„the manual waking movement “ for min. 2 sec.
Battery > 3.3V
no
Connect Battery
Rese t
PwrTmr = t1 (180s)
Connection Establishmen
Connection disconnecte d
PwrTmr=0 ?
Connection?
yes
yes
Acoustic signal
RdTmr = t3 (20s)
PwrTmr = t2 (60s)
Full Enterprise Mode
„%off“ Command
RdTmr=0 ?
yes
yes
yes
Standby Mode
„ manual waking movement“ or Characters sent?
PwrTmr=0 ?
yes
Sleep Mode

2 TagTrans-Befehle

Normalerweise 'routet' TagTrans Host-seitig ankommende Zeichen an das HF/UHF-Modul weiter.

Damit man Änderungen, Abfragen etc. des TagTrans vornehmen kann, wurde ein einfacher Syntax eingeführt.

2.1 Allgemeiner Syntax

Der Syntax beginnt mit einem '%'-Zeichen und endet mit einem 'Return' (0Dh, 0Ah). Die Zeichen sollen kleiner sein als 80h.

%Befehl	*[para1 para2 ...]*{0Dh, **0Ah**}
z.B. %help	

2.1.1 Command „%help“

Gibt eine Liste der möglichen Befehle aus.

Input: **%help**	Output instruction list
	Output: *%TagTrans HW 2.2 SW 0.12.4 HF 3AMS Build Time Apr 29 2010 11:48:13* *%get to ... get timeouts* *toff actual time until power off* *troff actual time until reader off* *t1 timeout power off* *t2 retrigger time power off* *t3 retrigger time reader off* *%set t1 xxxxx ... set timeout power off to xxxxx seconds* *%set t2 xxxxx ... set retrigger time power off to xxxxx seconds* *%set t3 xxxxx ... set retrigger time reader off to xxxxx seconds* *%set factdef [rf/tt/bt] ... set TagTrans to factory default* *%get bat/bth ... get battery level/thresholds* *%set pin xxxx ... set pin (max 16 digits)* *%set name xxxx ... add string to friendly name (max 32 digits)* *%get acon ... get state of auto connect* *%set acon on/off ... set auto connect on/off* *%del pair ... delete pairing* *%set msg on/off ... set message on/off* *%beep volume[0-3] frequency[10Hz] duration[0.1s]* *%blink help ... lists blink commands* *%get br ... get actual baudrate* *%set br xxx ... set baudrate to xxx* *47 -> 9600, 23 -> 19200, 11 -> 38400,* *7 -> 57600, 3 -> 115200* *%ax help ... lists 3axis motion sensor commands* *%off [nowake] ... power down*

2.2 Befehlseingabe

Durch das Host -seitige Senden eines '%'-Zeichens wird der Datenfluss unterbrochen. Jedes weitere Zeichen wird nun in ein Buffer gespeichert (max. 254 Zeichen), dabei wird auch ein **Zeitzähler** aktiviert und mit ca. **8 Sekunden** geladen. Jedes weitere Zeichen initialisiert den Zeitzähler mit 8 Sekunden. Wenn der **Zeitzähler abgelaufen** ist **oder** wenn ein **Zeichen größer als 07Fh** ist, wird dieser Modus unterbrochen und jedes gespeicherte Zeichen (inkl. '%'-Zeichen) an das HF-Modul weitergesendet.

Wird hingegen – innerhalb des Zeitraums – ein **Return** (0Dh, 0Ah) gesendet, wird die gespeicherte Zeichenkette als ein Befehl behandelt und der Zeitzähler deaktiviert.

Ist der Befehl ungültig, wird der Buffer gelöscht und anschließend die Zeichenkette „*%syntax error*{CrLf}“ an den Host zurückgesendet.

2.3 Hardware Befehle

Diese Firmware unterstützt alle Hardware-Kombinationen, die sich mit einigen Befehlen einstellen lassen.

2.3.1 Befehl „set rf“

Mit diesem Befehl kann man zwischen HF- und UHF-Modul wählen.

%set rf 1	Kommunikation HF Modul und TagTrans
%set rf 2	Kommunikation UHF Modul und TagTrans

2.3.2 Befehl „set img“

Falls ein Imager (PL4407) installiert ist, aktiviert man sämtliche zugehörige Funktionen mit folgendem Befehl.

%set img 1	Aktiviert die Imager-Funktionen
%set img 0	Deaktiviert die Imager-Funktionen

2.3.3 Befehl „set mb“

Die „MultiBlock“ Funktionalität wird mit diesem Befehl aktiviert.

%set mb 1	Aktiviert MultiBlock-Funktion
%set mb 0	Deaktiviert MultiBlock-Funktion

2.4 Power-Management Befehle

TagTrans bietet die Möglichkeit den Stromverbrauch optimal an den jeweiligen Einsatz entsprechend anzupassen. Dieser Abschnitt beschreibt alle erforderlichen Anweisungen zum Aktivieren von verschiedenen Optionen zur Energieverwaltung.

2.4.1 Befehl „set t1/t2/t3“

Mit diesem Befehl kann man die Werte ändern, mit denen die Zeitzähler **PwrTmr** und **RdTmr** geladen werden – siehe Kapitel Power Management.

%set t1 *x*	PowerOff Timeout beim Start setzen, x wird in Dezimal und in Sekunden angegeben
%set t2 *x*	PowerOff Timeout setzen, x wird in Dezimal und in Sekunden angegeben
%set t3 *x*	ReaderOff Timeout setzen, x wird in Dezimal und in Sekunden angegeben

2.4.2 Befehl „get to“

Liefert den aktuellen Stand der einzelnen Zeitzähler und die entsprechenden Werte zurück.

%get to	Zeigt den Time-out Status
	Ausgabe: toff: 'OffTimer' troff: 'ReaderOff Timer' t1: 't1' t2: 't2' t3: 't3'\r\n *toff: x troff: y t1: a t2: b t3: c* *x* Zeit bis zum Schlafmodus - PwrTmr *y* Zeit bis zum Bereitschaftsbetrieb - RdTmr *a,b,c* sind die entsprechenden Ladewerte t1, t2, t3.

2.4.3 Befehl „get bat“

Liefert den aktuellen Batteriestand zurück.

%get bat	Liefert den Batteriestatus
	Ausgabe: *%bat x* *x* Der Batteriestand errechnet sich dann, wie folgt: U.bat = *x* * 0,0245 [V]

2.4.4 Command „get bth“

Returns the battery thresholds.

%get bth	Show battery thresholds
	Output: *%bth* 'ths1' 'ths2' 'ths3'\r\n

2.4.5 Befehl „off“

Mit diesem Befehl wechselt TagTrans sofort in den **Schlafmodus**, das heißt dass die Verbindung zum Host getrennt wird und somit der TagTrans nur mehr manuell „geweckt“ werden kann.

%off	
	Ausgabe: *%power down*

2.4.6 Befehl „set demo“

Dieser Befehl wurde zum Zwecke der Präsentation eingeführt und bewirkt, dass der Modus **Verbindungsaufbau** entfällt – siehe Kapitel Power Management. Jedoch wird trotzdem im Hintergrund versucht eine Verbindung mit einem Host aufzubauen. Solange keine wirkliche Verbindung besteht, wird diese dem TagTrans „vorgetäuscht“.

%set demo 1	Aktiviert den Demo-Modus
%set demo 0	Deaktiviert den Demo-Modus

2.5 Bluetooth Befehle

Das Bluetooth Modul sorgt für die Verbindung zwischen Host und TagTrans. Es arbeitet selbstständig und kann nur über eine direkte Verbindung, also über TagTrans, konfiguriert werden.

Wenn man zum Beispiel die Verschlüsselung ändern will, muss zuvor eine Verbindung zum Host aufgebaut worden sein, um TagTrans den entsprechenden Befehl senden zu können.

Bei einem Zurücksetzen der Werkseinstellung werden die Bluetooth Einstellungen ebenfalls zurückgesetzt.

2.5.1 Befehl „set name“

Hiermit kann der Name festgelegt werden mit dem sich TagTrans in einem Bluetooth-Netzwerk anmelden soll.

%set name *str*	*str* ist eine Zeichenkette, die dem Bluetooth-Modul übergeben wird.

Falls ein Name nicht wünschenswert ist, sendet man statt der Zeichenkette ein 'Return' als erstes Zeichen.

2.5.2 Befehl „set pin“

Die Verbindung zwischen Host und TagTrans ist standardmäßig verschlüsselt mit „1234“. Folgender Befehl ermöglicht das Ändern des Schlüssels:

%set pin *str*	str ist eine Zeichenkette, die dem Bluetooth-Modul übergeben wird.

Falls eine Verschlüsselung nicht erforderlich ist, sendet man ein 'Return' als erstes Zeichen.

2.5.3 Befehl „set acon on / set acon off / get acon“

Sobald der TagTrans im Betrieb ist, versucht standardmäßig das Bluetooth-Modul fortlaufend eine Verbindung aufzubauen. Mit diesem Befehl kann diese Funktion – automatische Verbindung – aktiviert bzw. deaktiviert werden.

%set acon on	Aktiviert die Automatische Verbindung
%set acon off	Deaktiviert die Automatische Verbindung
%get acon	Liefert den aktuellen Stand zurück

2.5.4 Befehl „sendbt“

%sendbt	Sendet iWRAP Protokoll Befehle zum WT11 Modul
	Ausgabe: %ok\r\n

2.5.5 Befehl „del pair“

Falls TagTrans bereits mit einem Host verbunden ist, können die beiden gepaart werden, das heißt dass die Adressen des „Partners“ gespeichert werden. Um diese paarweise Verbindung zu unterbinden, kann folgender Befehl verwendet werden:

%del pair	Löst die paarweise Verbindung auf

2.5.6 Befehl „set factdef bt“

%set factdef bt	Setzt das BT Modul auf seine Herstellereinstellungen zurück

2.6 HF und UHF Modul Befehle

Die Kommunikation zwischen HF-Modul bzw. UHF-Modul und TagTrans erfolgt seriell. Manchmal ist es nötig, die Kommunikationseinstellungen anzupassen.

2.6.1 Befehl „set/get br“

Mit diesem Befehl kann man die Baudrate, mit der TagTrans empfangen bzw. senden soll, einstellen.

%set br *x*	*x*	3 …115200
	x	7 …57600
	x	11 …38400
	x	23 …19200
	x	47 …9600
%get br	Liefert den aktuellen Wert zurück. Ausgabe: %br x	

2.6.2 Befehl „poll [on/off]“ (nur für UHF Module)

Um eine kontinuierliche Abfrage der Seriennummer der UHF-Tags zu gewährleisten, sendet TagTrans ein „polling“ - Befehl an das UHF-Modul. Da im Bereitschaftsbetrieb das UHF Modul nicht versorgt wird, muss dieser Befehl bei jedem Start des Vollbetriebs gesendet werden. Um diesen Vorgang zu automatisieren, bedient man sich den optionalen Parametern „on“ bzw. „off“.

%poll	Sendet „polling“-Befehl an das UHF-Modul
%poll on	Automatisches Senden des „polling“-Befehls wird angewendet
%poll off	Automatisches Senden des „polling“-Befehls wird *nicht* angewendet
	Output: %OK\r\n Befehl wurde erfolgreich versendet %poll 1\r\n %poll 0\r\n *%no response* UHF-Modul reagiert nicht

2.6.3 Befehl „**mb** [*cmd*]/[set/get auto/key/tag]“ / „**md** [*cmd*]“ / „**mt** [*cmd*]“

„mb“ steht für MultiBlock und dient unter anderem der formatierten Ausgabe von Speicherinhalten eines Tags. Der Parameter **„[*cmd*]“** bildet hierfür die **Abfolge** (oder Befehlskette), mit der TagTrans z.B. einen Tag auslesen soll – siehe Kapitel MultiBlock

Es besteht die Möglichkeit diese Abfolge zu automatisieren. Hierbei wird das HF-Modul in den sogenannten „continuous-mode“ betrieben und die Anzahl gleicher eingehender UIDs gezählt. Erreicht diese Anzahl einen Wert von **3**, wird die gespeicherte Abfolge ausgeführt. Hierbei sollte man beachten, dass das HF-Modul in diesem Modus **Host-seitig nicht ansteuerbar** ist – es werden auch keine Zeichen von HF-Modul an den Host direkt weitergeleitet.

Für manche Befehle innerhalb der Abfolge ist es nötig einige Parameter, der anzuwendenden Transponder, den TagTrans zu übermitteln – Die **Länge eines Blocks** in Byte bzw. **der erste und der letzte Block**. Eigens erstellte Abfolgen können mit dem „**md** ...“-Befehl (MultiBlock **Debug**) auf ihre Richtigkeit überprüft werden. Die Messung bzw. Ausgabe der benötigten Rechenzeiten für eine Abfolge erfolgt mittels „**mt** ...“-Befehl (MultiBlock **Timing**).

%mb	Startet die **gespeicherte** Abfolge
%mb [cmd]	Startet die **übergebene** Abfolge Ausgabe: *{laut der Abfolge}*
%md	Startet die **gespeicherte** Abfolge und gibt **Debugging-Informationen** aus.
%md [cmd]	Startet die **übergebene** Abfolge und gibt **Debugging-Informationen** aus Ausgabe: *Pos func id in/out iflevel sub* *….* *exit code: ddd @pos: hh. t.all: t.tts t.calc: t.tts* *ddd* – Fehlercode - siehe Tabelle *hh* – Letzte Position der Abfolge t.all *t.tt* - gesamte Zeitaufwand (inkl. Ausgabe) in Sekunden t.calc *t.tt* - reine Rechen-/Wartezeit in Sekunden
%mt	Startet die **gespeicherte** Abfolge und gibt **Timing-Informationen** bzw. Anzahl diverser Kommunikationsfehler aus.
%mt [cmd]	Startet die **übergebene** Abfolge und gibt **Timing-Informationen** bzw. Anzahl diverser Kommunikationsfehler aus. Ausgabe: *{according the sequence}* *t.all: t.tts Err:?dd;Cdd;Fdd;Idd;Ndd;Odd;Rdd;Xdd;* *dd* - ist die Anzahl der Fehler in Dezimal *t.tt* - gesamte Zeitaufwand (inkl. Ausgabe) in Sekunden
%ms [cmd]	Speichert die eingegebene Abfolge, ohne diese auszuführen.
%mb set key	Speichert die zuletzt ausgeführte Abfolge Keine Ausgabe
%mb get key	Gibt die gespeicherte Abfolge aus Ausgabe: *[cmd]*
%mb set tag *ll ss ee*	Übergibt TagTrans die Parameter der anzuwendenden Tags *ll* – die Länge eines Blocks in Byte in Dezimal *ss* – der erste Block in Dezimal (od. Hexadezimal mit Hss) *ee* – der letzte Block in Dezimal (od. Hexadezimal mit Hee) Keine Ausgabe.
%mb get tag	Gibt die gespeicherten Parameter aus Ausgabe: *%mb len:ll first:ss last:ee*
%mb set auto *x*	Automatisches Ausführen der Abfolge einstellen 0=off / 1=on Keine Ausgabe
%mb get auto	Gibt die aktuelle Einstellung aus. Ausgabe: %mb auto=x 0=off / 1=on
%ms	Store MB command as key 0=off / 1=on Keine Ausgabe
%mb set cal	Disable/Enable MB Antenna calibration mode Keine Ausgabe

2.7 Imager Befehle

Um den Imager leichter in eine Anwendung zu integrieren, stellt TagTrans einige Funktionen bzw. Befehle zu Verfügung.

2.7.1 Befehl „set img“

Falls ein Imager (PL4407) installiert ist, aktiviert man sämtliche zugehörige Funktionen mit folgendem Befehl.

%set img 1	Aktiviert die Imager-Funktionen
%set img 0	Deaktiviert die Imager-Funktionen

2.7.2 Befehl „ic help“

Gibt eine Liste der möglichen Befehle aus.

%ic help	Befehlsliste ausgeben Ausgabe: *%ic [init/off] ..init imager / turn off imager* *%ic wakeup [on/off/trig] ..set/trigger wakeup* *%ic set/get prefix/suffix ..use ascii or \xx(hex), \r, \n, * *%ic hc start ..direct communication mode* *%ic hc set/get to [sec] ..set/get hc-timeout (2s..200s)* *%ic hc init [on/off] ..set hc on startup* *%ic toggle [/off/rf/img] ..start/toggle* *%ic toggle set/get to [dsec] ..toggle timeout (2ds..100ds)* *%ic toggle init [on/off] ..togglemode on startup* *%ic msg [on/off] ..set imager message on/off* *%ic get pin ..get hardware pin* *%ic 0/1 ..start/Stopp dec* *%ic 2/3 ..aim off/on* *%ic 4/5 ..led off/on* *%ic 6 ..sleep* *%ic 7..set default* *%ic 8 pp aa [bb cc] (hex)..param send* *%ic 9 pp aa [bb cc] (hex)..param send&store* *%ic 10 pp (hex)..param request*

2.7.3 Befehl „ic init“

Initialisiert das Imager-Modul. Dies geschieht automatisch bei jedem Reset-Vorgang.

%ic init	Initialisierung des Imagers Ausgabe: *%init OK*

2.7.4 Befehl „ic off“

Das Imager-Modul wird nicht mehr versorgt. Die Imager-Befehle sind jedoch zugänglich.

%ic off	Imager-Modul deaktivieren
	Keine Ausgabe

2.7.5 Befehl „ic *x*“

Um den Endbenutzer die Kommunikation mit den Imager (SSI-Protokoll) zu ersparen, wurden die wichtigsten Befehle mit diesem Befehl vereinfacht.

***%ic* 0**	Startet den Dekodiervorgang
%ic 1	Stoppt den Dekodiervorgang
%ic 2	Aktiviert das Fadenkreuz
%ic 3	Deaktiviert das Fadenkreuz
%ic 4	Aktiviert die LED (wenn vorhanden).
%ic 5	Deaktiviert die LED
%ic 6	Aktiviert den Sleep Mode

Sämtliche Einstellungen des Imager - Moduls können mit den folgenden Befehlen abgefragt bzw. verändert werden. In der *SE4407-Dokumentation* sind alle Parameter und deren Werte in Detail beschrieben.

%ic 8	Ändern eines Parameters (in Hexadezimal) (wird nicht gespeichert!)
%ic 10	Abfragen eines Parameters (in Heaxadezimal)

Die folgenden Befehle sollten nicht kontinuierlich (z.B. in einer Schleife) ausgeführt werden:

%ic 7	Einstellungen des Imager -Moduls rücksetzen
%ic 9	Ändern und speichern eines Parameters (in Hexadezimal)
	Ausgabe: „ic 10 xx“ *%para=xx val=yy* *xx* – Parameter *yy* – zugehöriger Wert (siehe SE4407 Dokumentation)

Errorliste Ausgabe

%ic error 'ic_err'\r\n	
%ic error **1** \r\n	Timeout
%ic error **2** \r\n	Allgemeiner Fehler
%ic error **3** \r\n	Keine Rückmeldung
%ic error **4** \r\n	Imager nicht aktiviert
%ic error **5** \r\n	Imager nicht bereit oder nicht angeschaltet
%ic error **6** \r\n	Syntax Fehler
%ic error **7** \r\n	Empfangspuffer Fehler
%ic error **8** \r\n	Checksummen Fehler
%ic error **9** \r\n	Falscher Parameter

2.7.5.1 Beispiel

„%ic 8 E3 01“	… MicroPDF417 – Erkennung aktivieren (nicht speichern).
„%ic 0“	… Startet den Decodevorgang (Timeout lt. Einstellung Imager - Modul)
„%ic 1“	… Decodevorgang stoppen.
„%ic 3“	… Fadenkreuz ausschalten.
„%ic 6“	… In den Sleepmodus gehen.

2.7.6 Befehl „ic wakeup on/ ic wakeup off/ ic wakeup trig“

Über diesen Befehl wird der WakeUp-Eingang des Imager-Moduls angesteuert.

„ic wakeup on“	Sleepmodus wird deaktiviert
„ic wakeup off“	Automatischer Sleepmodus aktiviert
„ic wakeup trig“	Imager - Modul kurz aktivieren.
	Keine Ausgabe

2.7.7 Befehl „ic set prefix/ ic set suffix/ ic get prefix /ic get suffix “

Nach einem erfolgreichen Decode-Vorgang werden die Daten direkt in ASCII übertragen. Um solch einen Decode-Vorgang erkennbar zu machen, können Präfix- bzw. Suffix-Zeichenketten angefügt werden.

%ic set prefix *text*	Präfix verändern.
%ic get prefix *text*	Präfix ausgeben.
%ic set suffix *text*	Suffix verändern.
%ic get suffix *text*	Suffix ausgeben.
	Ausgabe: Bei Parameter „get …“ wird die jeweiligen Zeichenkette ausgegeben.

Beachten:

Spezielle Zeichen können mit einem Backslash angegeben werden.
Erkannt werden: „\r“, „\n“, „\\“, „\00“, wobei die Zahl zweistellig und in Hexadezimal sein muss. Die maximale Länge: 10 Zeichen.

2.7.7.1 Beispiel

„%ic set prefix \FFdecoder:“	… Präfixtext setzen -> {0xFF}decoder:
„%ic set suffix end\r\n”	… Suffixtext setzen -> end{CrLf}

2.7.8 Befehl „ic hc start / ic hc init on / ic hc init off / ic hc set to / ic hc get to“

Mit diesem Befehl wird die direkte (binäre) Kommunikation mit dem Imager - Modul ermöglicht. Nach einem einstellbaren Timeout wird die direkte Kommunikation abgebrochen.

%ic hc start	Startet die direkte binäre Kommunikation.
%ic hc init on	Startet die binäre Kommunikation beim Startup. Automatische Ausführung des Start-Befehls nach einem WakeUp.
%ic hc init off	Startet nicht die binäre Kommunikation beim Startup. Keine automatische Ausführung des Start-Befehls nach einem WakeUp.
%ic hc set to *x*	Setzen des Timeout für die binäre Kommunikation = *x* (in Sekunden; max. 100)
%ic hc get to	Abfragen des Timeout für die binäre Kommunikation
%ic hc to off	Aktivieren des Timeout für die binäre Kommunikation
%ic hc to on	Deaktivieren des Timeout für die binäre Kommunikation
	Ausgabe: „%ic hc get to“: *%hc timeout: x s* *x* – time in Sekunden

2.7.9 Befehl „ic toggle/ ic toggle init on/ ic toggle init off / ic toggle set to ic toggle get to/ ic toggle img / ic toggle rf/ ic toggle off“

Dieser Befehl bringt nach einem einstellbaren Timeout das Imager -Modul in den Sleep -Modus bzw. startet den Decode -Vorgang. Sobald ein Zeichen übertragen wird, wird dieser Timeout zurückgesetzt. Falls ein erfolgreicher Decode -Vorgang vom Imager -Modul erkannt wurde, wird das Ergebnis, wie mit dem Befehl „%ic 0“, an den Host übertragen und das Modul sofort in den Sleep -Modus gesetzt.

%ic toggle	aktiviert den Toggle –Modus / wechselt zwischen „Sleep -Modus“ und „Active -Modus“
%ic toggle img	wechselt zu “Active –Modus”
%ic toggle rf	wechselt zu RFID Lesen und in “Sleep -Modus”
%ic toggle off	deaktiviert den Toggle –Modus
%ic toggle set to *x*	setzt den Timeout = *x* (in 0.1s)
%ic toggle get to	gibt den Timeout (in 0.1s) aus
	Ausgabe: %toggle timeout: 0 ds %toggle init: off Bei dem Parameter „get to“ wird der Timeout in 0.1s ausgegeben.
%ic toggle init on	Toggle -Modus nach einem WakeUp
%ic toggle init off	kein Toggle -Modus nach einem WakeUp
	Ausgabe: Ausgabe wie bei „ic 0“.

2.7.9.1 Beispiel

%ic toggle set to 8	setzt den Timeout auf 0.8s
%ic toggle	aktiviert den Toggle-Modus
%ic toggle rf	wechselt zu „sleep-Modus“
%ic toggle off	deaktiviert den Toggle-Modus

2.7.10 Befehl „ic msg on/off“

Aktivierung/Deaktivierung der Fehlerausgabe.

%ic msg on	Fehlerausgabe aktiviert.
%ic msg off	Fehlerausgabe deaktiviert.
	Keine Ausgabe

2.7.11 Befehl „ic get pin“

Gibt die aktuellen Zustände jeder einzelnen Pins des Moduls bzw. des TagTrans aus.

%ic get pin	Ausgabe der Pin-Zustände
	Ausgabe: %ic vcc=0 %ic aim/wakeup=1 %ic pwrdwn=0 %ic decode=0 %ic trigger=1 %ic rts=0 %ic cts=1 %ic download=1 x – binary Wert (0 od. 1)

2.8 Bewegungssensor - Befehle

Der 3 Achsen Bewegungssensor ermittelt Bewegungen bzw. Beschleunigungen jeder Richtung. Grundsätzlich können 2 Bewegungsarten automatisch erkannt werden:

„der freie Fall“
z.B. Schwenken des TagTrans von unten nach oben od. von oben nach unten.

„Schlag od. Stoß“ (einfach od. zweifach)
z.B. ein doppeltes „Antippen“ des TagTrans mit dem Daumen innerhalb einer halben Sekunde.
Oder
z.B. eine 360° Drehbewegung um die Längsachse innerhalb einer zehntel Sekunde.

Diese Bewegungen können mit einer der folgenden **Aktionen (Events)** beliebig kombiniert werden:

„Aufwecken des TagTrans“
„Alle Zeitzähler zurücksetzten“
„TagTrans in Schlafmodus versetzen“
„Ausführen des gespeicherten MultiBlock Commands“

Es gibt 2 verschiedene Konfigurationsmodi:

„**Aktiver Modus“**
Szenario: eine Bewegung wird erkannt und eine Aktion (Event) wird ausgeführt.
„**WakeUp Modus“**
Szenario: eine Bewegung wird erkannt und TagTrans wird aus dem Schlafmodus geweckt.

Hinweis:
Alle Schwellwerte beziehen sich auf das x-fache der Erdbeschleunigung g und können zwischen 0 und 15 sein.
Der Wert 1 entspricht in etwa 0.5*g mit einem relativen Fehler von 15%.

2.8.1 Befehl „axr”

Der Sensor enthält ein 8-Bit-Register, das verwendet wird, um sein Verhalten zu kontrollieren bzw. um Beschleunigungsdaten abzurufen. Die Register-Adresse besteht aus 7 Bit.

Mit dem Befehl „%axr“ kann das Register ausgelesen werden.

%axr	Register auslesen
	Ausgabe: 'reg_adress'->'reg_value'\r\n

2.8.2 Befehl „axw”

Mit dem Befehl „%axw“ kann das Register beschrieben werden. (Achtung nicht für den Anwender zu empfehlen!)

%axw	Register beschreiben
	Ausgabe: 'reg_adress'<-'reg_value'\r\n
	Fehlerausgabe: %ax error\r\n %ax error com\r\n

2.8.3 Befehl „ax help”

Listet alle 3 Achsen Motion Sensor Befehle auf.

%ax help	Listet Sensor Befehle auf.

	Ausgabe: 3axis motion sensor command list %ax config wakeup/active ... select wakeup/active config %ax get config ... show configuration %ax set event C X N ... select event n after x motion c=0 -> channel 1 c=1 -> channel 2 x=0 -> no n=0 -> no event/wake up x=1 -> free fall n=1 -> turn off TagTrans x=2 -> click n=2 -> retrigger timer n=3 -> execute MultiBlock %ax filter [val] ... set high-pass filter [0..4] Free fall detect settings... %ax ff ths [val] ... set threshold 2..15 %ax ff [x/y/z][0/</>] ... start event if 0 -> ignore axe x/y/z > -> active if accel. of axe x/y/z is greater than ths < -> active if accel. of axe x/y/z is smaller than ths %ax ff dur [csec] ... set free fall duration 0..255csec Click detect settings... %ax clk [x/y/z][0/1/2/3] ... none/single/double/both clicks %ax clk ths [x/y/z] [val] ... set threshold 2..15 %ax clk tlim [msec] ... set time limit for one click 0..127msec %ax clk tlat [msec] ... set latency after click 0..255msec %ax clk twin [msec] ... set time window for two clicks 0..255msec %ax chk ... starts testing mode

2.8.4 Befehl „ax init”

Der Sensor wird initialisiert.

%ax init	Initialisierung des Sensors
	Ausgabe: %ax ok\r\n

2.8.5 Befehl „ax stop”

Der Sensor wird ausgeschalten – Sleep Modus. (Damit der Energieverbrauch reduziert wird.)

%ax stop	Sensor Stopp
	Ausgabe: %ax ok\r\n

2.8.6 Befehl „ax config wakeup”

Mit diesem Befehl wird TagTrans in den WakeUp-Konfigurationsmodus versetzt.

%ax config wakeup	Konfiguration Wakeup Modus
	Keine Ausgabe

2.8.7 Befehl „ax config active”

Mit diesem Befehl wird TagTrans in den Active-Konfigurationsmodus versetzt – standardmäßig nach jedem Neustart.

%ax config active	Konfiguration Active Modus
	Keine Ausgabe

2.8.8 Befehl „ax get config”

Zeigt den derzeitigen Konfigurationsmodus und listet die zugerhörige Konfiguration auf.

%ax get config	
	Ausgabe: Configuration mode: active Channel 1: retrigger timer if click Channel 2: no event/wake up if none High-pass filter cut-off freq.: 2Hz Free fall settings: Threshold: 2 Duration: 10 csec Axis: X> Y> Z> Click settings: Threshold: X=2 Y=2 Z=2 Time limit: 32 msec Time latency: 100 msec Time window: 100 msec Axis: X both, Y both, Z both

2.8.8.1 Beispiel

Eingabe:	
%ax config wakeup **%ax get config**	
	Ausgabe: Configuration mode: wakeup Channel 1: no event/wake up if click Channel 2: no event/wake up if none High-pass filter cut-off freq.: 2Hz Free fall settings: Threshold: 2 Duration: 10 csec Axis: X> Y> Z> Click settings: Threshold: X=2 Y=2 Z=2 Time limit: 32 msec Time latency: 100 msec Time window: 100 msec Axis: X both, Y both, Z both

2.8.9 Befehl „ax set event”

Mit diesem Befehl kombiniert man automatisch erkannte Bewegungen mit einer beliebigen Aktion. Es können simultan zwei verschiedene Bewegungen erkannt werden (Channel1 / Channel 2)

Dieser Befehl hat folgende Syntax:

%ax set event *a b c*

a … **0 oder 1** -> 1. oder 2. Bewegungserkennung

b … **0 bis 2** -> 0 = keine Bewegung, 1 = freier Fall, 2 = Schlag/Stoß

c … **0 bis 3** -> 0 = Aufwecken, 1 = Zeitzähler zurücksetzten, 2 = TagTrans auschalten, 3 = MultiBlock ausführen

%ax set event	Ereignis Trigger setzen (event <-> trigger)
	Keine Ausgabe

2.8.10 Befehl „ax filter”

Der Hochpassfilter wird gesetzt um statische Beschleunigungen herauszufiltern (z.B. Erdbeschleunigung).

Dieser Befehl hat folgende Syntax

%ax set event *a*

b … **0 bis 4** -> 0 = statisch, 1 = 0,25 Hz, 2 = 0,5 Hz, 3 = 1 Hz, 4 = 2 Hz

%ax filter	Hochpassfilter setzen
	Keine Ausgabe
	Fehlerausgabe: %ax error\r\n

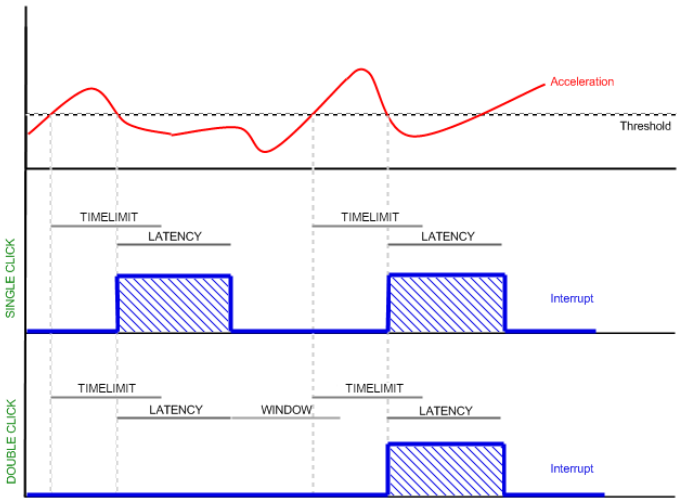
2.8.11 Befehl „ ax ff / ax ff ths / ax ff dur”

Für den freien Fall müssen gesetzt werden: Achse, Schwelle und Dauer.

%ax ff	Axis - Setzen der Achse für den freien Fall (größer oder kleiner als die Schwelle)
%ax ff ths	Threshold - Die Schwelle des freien Falles festsetzen
%ax ff dur	Duration - Die Dauer innerhalb od. außerhalb der Schwelle des freien Falles festsetzen
	Keine Ausgabe
	Fehlerausgabe: %ax error\r\n

2.8.12 Befehl „ax clk / ax clk ths / ax clk tlat / ax clk tlim / ax clk twin“

Der Sensor kann zwischen zwei verschiedenen Schlag-/Stoßarten unterscheiden – einfach od. zweifach.



In der Abbildung werden ein doppelter Stoß und ein einfacher Stoß abgebildet.

%ax clk	Setzten der Achse x, y oder z
%ax clk ths	Die Schwelle für x/y/z Achse setzen – jede Achse kann individuell gesetzt werden.
%ax clk tlat	Verzögerung der Aktion nach der Bewegungserkennung.
%ax clk tlim	Zeitlimit für die Schlag-/Stoßerkennung setzen
%ax clk twin	Maximal zulässiges Zeitfenster für zweifachen Schlag/Stoß setzen
	Keine Ausgabe
	Fehlerausgabe: %ax error\r\n

2.8.13 Befehl „ax chk”

Zum Test Modus wechseln - dient in erster Linie zur Überprüfung der eingestellten Bewegungserkennung. Für die erste Bewegungserkennung ertönt ein tiefes Signal und für die zweite ein hohes. Zusätzlich werden alle 120 Millisekunden die aktuellen Beschleunigungswerte der Achsen angezeigt. Sie können hier z.B. die ausgegebenen Daten über ein Terminalprogramm in eine Datei speichern und weiter verarbeiten.

%ax chk	Wechsel zum Testmode
	Ausgabe: %'output for 3axis'\r

2.8.13.1 Beispiel

Eingabe:	
%ax chk	
	Ausgabe: %3ax testing mode %type 'h' for help

2.8.14 Befehl „ax set default”

Den Sensor auf Herstellereinstellungen zurücksetzten.

%ax set default	Set Motion Sensor to factory default
	Ausgabe:

	%ax ok\r\n

2.9 MultiBlock

In diesem Kapitel wird die Funktionsweise der MultiBlock-Abfolge bzw. die möglichen Befehle innerhalb einer Abfolge erklärt.

2.9.1 Dekodierung der Abfolge

- Die Befehlserkennung erfolgt Zeichenweise.
- Wird ein Befehl erkannt, wird dieser sofort ausgeführt und die Ausgabe erfolgt unmittelbar.
- Mit dem Zeichen **„.“** wird eine Abfolge beendet.
- Ein **{return}-Steuerzeichen** bricht, um ein Überlauf zu vermeiden, die Abfolge ab.
- Jeder Fehler, der während der Befehlserkennung entsteht, führt zu einem Abbruch. Eine Ausnahme stellt die „If-Anweisung“ dar - jedoch nur wenn kein Syntax-Fehler vorliegt.
- Die maximale Länge einer Abfolge beträgt 250 Zeichen.
- Es wird zwischen Groß- und Kleinschreibung unterschieden.

2.9.2 Speicherverwaltung

- Jeder **Block**, der **gelesen oder beschrieben** wird, wird in einem **statischen Speicher** abgelegt.
- Falls ein, bereits abgespeicherter, Block gelesen werden soll, wird dieser vom Speicher ausgelesen.
- Bei einem erneuten Start der Abfolge, wird der Speicher gelöscht.

2.9.3 Kommunikation mit HF-Modul

- Bei jedem Start einer Abfolge, wird die erste empfangene UID eines Tags „festgehalten“.
- Falls eine andere UID während einer Abfolge erkannt wird, führt dies zu einem Kommunikationsfehler. Somit wird das fälschliche Lesen oder Schreiben eines anderen Tags vermieden. Falls erwünscht, bietet ein Befehl 'Release Tag' die Möglichkeit den Tag „los zu lassen“. Danach wäre eine neue UID in derselben Abfolge möglich.
- Falls ein Block teilweise beschrieben werden soll, wird dieser vorher gelesen. (R&W)
- Einige Kommunikationsfehler führen zum sofortigen Abbruch, andere wider um erlauben mehrmalige Versuche.
- Die Kommunikation erfolgt über ASCII Zeichen.

2.9.4 Befehlsliste

Die Tabelle zeigt alle möglichen Befehle, die **innerhalb einer Abfolge** verwendet werden können.

Befehl	Befehls-name	Erklärung	Chapter	Parameter	Kondition	Textspeicher
"{text}**"**	Text	Gibt den {text} aus.	2.9.4.1	-	Ja	-
U	UID	Übergibt die aktuelle UID des Tags	2.9.4.2	Ja	Ja	Ja
R	Read	Liest Blockweise den Tag aus.	2.9.4.3	Ja	Ja	Ja
W	Write	Schreibt Blockweise auf den Tag.	2.9.4.4	Ja	Ja	-
PR	Pointer Read	Liest Byte weise den Tag aus.	2.9.4.5	Ja	Ja	Ja
PW	Pointer Write	Schreibt Byte weise auf den Tag.	2.9.4.6	Ja	Ja	-
T	Trigger	Setzt den „Reader-Timeout-Zähler“ zurück.	0	-	-	-
Q	Release Tag	Erzwingt das Entfernen des Tags vom Feld.	2.9.4.7	Ja	Ja	-
b	Beep	Gibt ein akustisches Signal aus.	2.9.4.8	Ja	-	-
I	Imager Decode	Sendet den Imager einen Decode-Befehl.	2.9.4.9	Ja	Ja	Ja
? : / ;	If, Then, Else, End If	Überprüft eine Bedingung: [...]?{Kondition}:{wahr}/{falsch};[...]	2.9.4.10	-	-	-
.	End	Beendet die Abfolge.	2.9.4.11	-	-	-

2.9.4.1 Text – Befehl

Ermöglicht das Senden von Zeichenketten an den Host. Spezielle ASCII Zeichen können mit einem Backslash '\' und einer darauf folgenden Zahl übergeben werden.

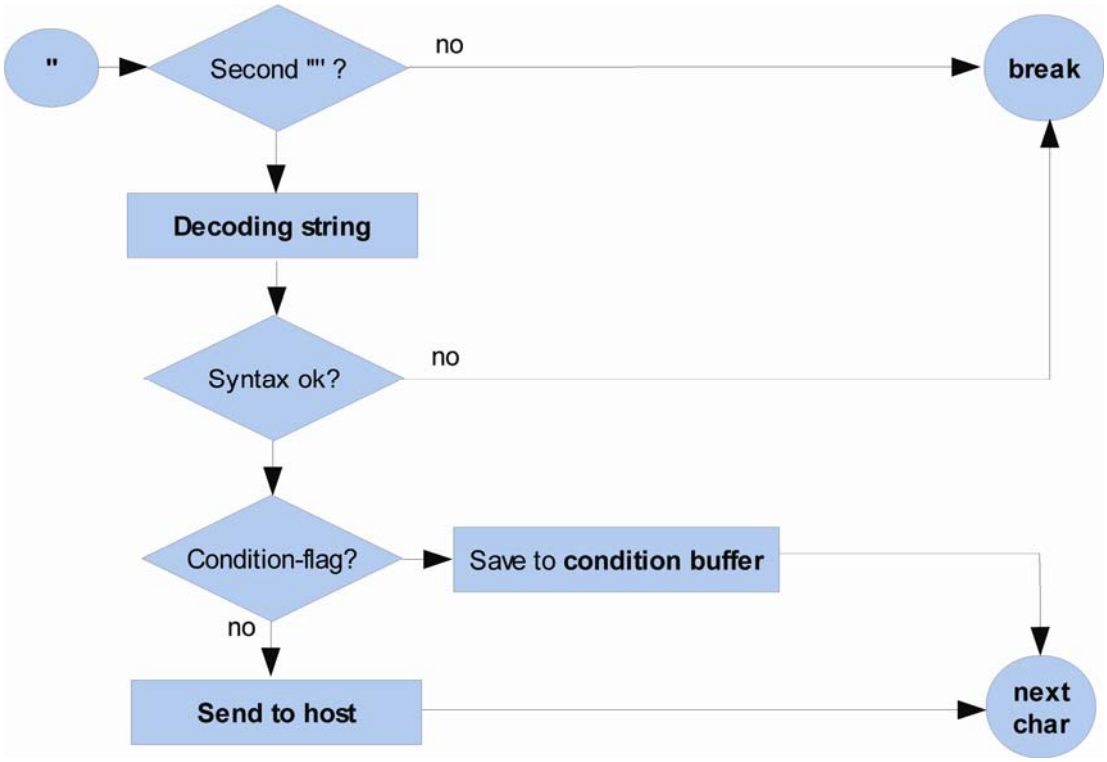
2.9.4.1.1 Syntax

"{text}"

2.9.4.1.2 Mögliche Sonderzeichen

\Hxx	xx = Hexadezimalzahl von 00..FF
\xxx	xxx = Dezimalzahl von 000..255
\r	Return
\n	neue Zeile
\t	Tabulator
\"	Anführungszeichen
\\	Backslash
\x[]	Textspeicher einfügen

2.9.4.1.3 Flussdiagramm



2.9.4.2 UID – Befehl

Sendet oder speichert die UID eines Tags. Die UID selbst wird von dem HF-Modul als Nibble ausgegeben.

2.9.4.2.1 Syntax

U[]	Gesamte UID
U[*a*]	Das *a*. Zeichen aus der UID (von 0 weg)
U[*a*-*b*]	Der *a*. bis *b*. Zeichen aus der UID (von 0 weg)
U[x]	Ausgabe wird im Textspeicher gespeichert

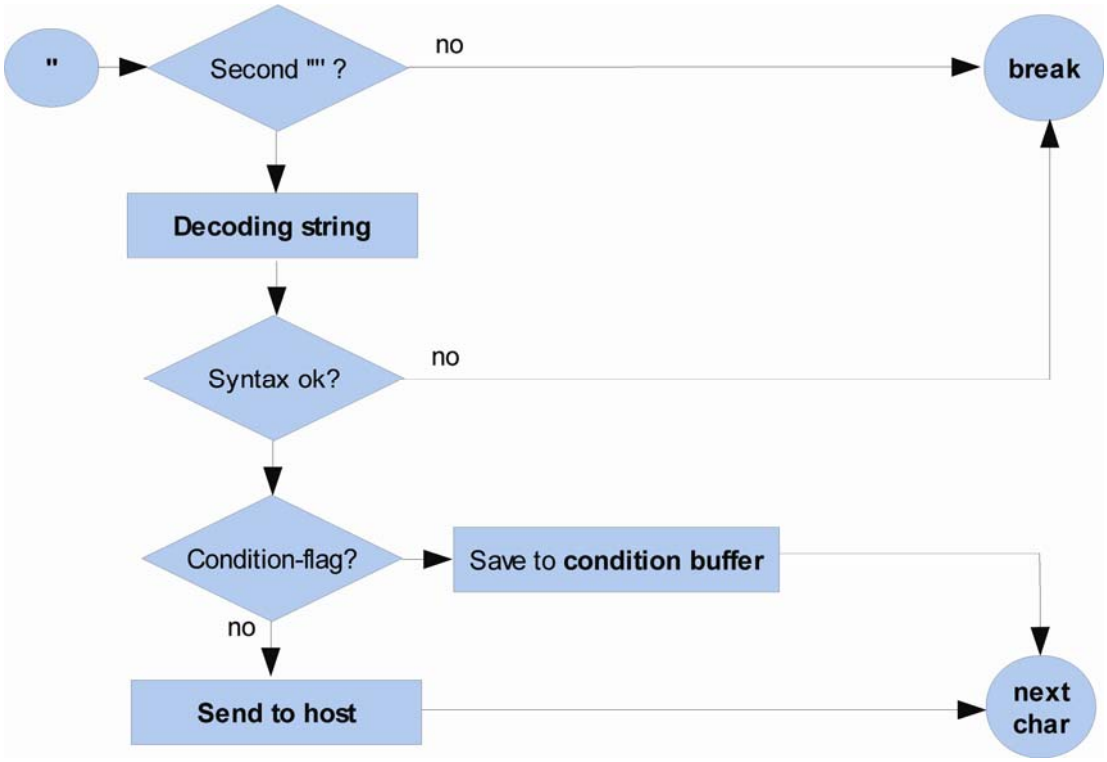
2.9.4.2.2 Erklärung

1. Parameter wird auf Syntax und Gültigkeit überprüft und gespeichert.
2. UID wird gelesen, falls noch nicht gelesen wurde.
3. Tritt ein Kommunikationsfehler auf (z.B. kein Tag im Feld) gibt es 2 Möglichkeiten:

   1. Befindet sich der Befehl innerhalb einer Kondition, wird der Fehlercode in Form von „err*xxx*“, wobei *xxx* gleich dem Fehlercode ist, als Zeichenkette gespeichert. Der Success-Flag wird auf „Falsch“ gesetzt.

   2. Befindet sich der Befehl außerhalb einer Kondition, wird die Abfolge abgebrochen.
4. Die UID Zeichenkette wird entsprechend dem Parameter manipuliert.
5. Ist der UID Befehl innerhalb einer Kondition, wird die Zeichenkette gespeichert und der Success-Flag auf „Wahr“ gesetzt.
6. Ist der UID Befehl außerhalb einer Kondition, wird die Zeichenkette direkt ausgegeben.

2.9.4.2.3 Flussdiagramm



2.9.4.3 Blockweise Lesen

Sendet bzw. speichert einen oder mehrere Blöcke eines Tags, wobei die Art der **Ausgabe** verschieden sein kann.

2.9.4.3.1 Syntax

Ausgabe in Byte	**Ausgabe in Nibble**	**Ausgabe in Dezimal** (16 Bit)
R[]	R{}	R()
R[*a*]	R{*a*}	R(*a*)
R[*a*-*b*]	R{*a*-*b*}	R(*a*-*b*)
R[*a*-*b*,*c*,*d*,*e*,*f*]	R{*a*-*b*,*c*,*d*,*e*,*f*}	R(*a*-*b*,*c*,*d*)
R[*a*-*b*,"*{text}*"]	R{*a*-*b*,"*{text}*"}	R(*a*-*b*,"*{text}*")
R[*a*-*b*,*c*,*d*,*e*,*f*,"*{text}*"]	R{*a*-*b*,*c*,*d*,*e*,*f*,"*{text}*"}	R(*a*-*b*,*c*,*d*,"*{text}*")

2.9.4.3.2 Parametererklärung

Parameter	**Erklärung**
leer	Lesen des 0. Blocks
a	Lesen des **a.** Blocks
a-b	Lesen des **a. bis b.** Blocks
c	**c.** Byte als **erstes** ausgeben; **wenn nicht angegeben: alle Bytes ausgeben** (bei Dezimal die ersten zwei Bytes)
d	**d.** Byte als **zweites** ausgeben; (bei Dezimal das **x.** Byte als **Low-Byte** und das **w.** Byte als **High-Byte** behandeln) wenn nicht angegeben: w. Byte ausgeben. (bei Dezimal das **w.** Byte als **Low-Byte** behandeln)
e	**e.** Byte als **drittes** ausgeben; wenn nicht angegeben: w. bis x. Byte ausgeben.
f	**f.** Byte als **viertes** ausgeben; wenn nicht angegeben: w. bis y. Byte ausgeben.
"*{text}*"	Trennt jeden Block mit der Zeichenkette **{text}** Es gelten die gleichen Regeln wie bei dem **Text-Befehl -** siehe Kapitel 2.9.4.1 Zusätzlich kann die **Textposition** der einzelnen - manipulierten - Blöcke und die **Blocknummer** ausgegeben werden. **&T** ... Textpositionsmarke **&C** oder **&C%%** … Positionsmarke Blocknummer in Hexadezimal **&c** oder **&c%%** oder **&c%%%** … Positionsmarke Blocknummer in Dezimal **&&** entspricht dem Zeichen **&** z.B.: "Blocknr: &C%% && content: &T \r\n" → Blocknr: 0A & content: 00FF01FE

2.9.4.3.3 Erklärung

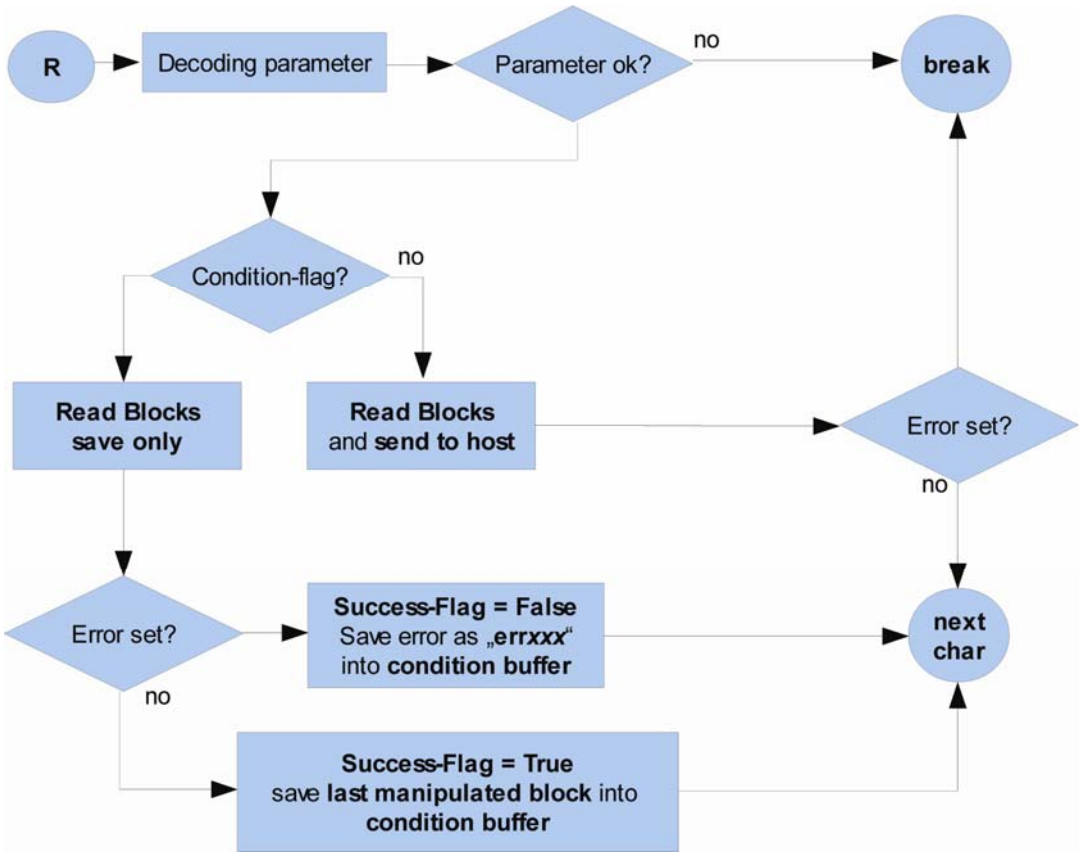
1. Parameter wird auf Syntax und Gültigkeit überprüft und gespeichert.
2. Ist der Befehl **innerhalb einer Kondition** wird die Funktion **'Read Block' ohne Ausgabe** aufgerufen. Ist er **außerhalb einer Kondition** wird die Funktion **mit Ausgabe** aufgerufen.
3. Tritt ein Kommunikationsfehler auf (z.B. Fehlerüberschreitung) gibt es 2 Möglichkeiten:

   1. Befindet sich der Befehl innerhalb einer Kondition, wird der Fehlercode in Form von „err*xxx*", wobei *xxx* gleich dem Fehlercode ist, als Zeichenkette gespeichert. Der **Success-Flag** wird auf **„Falsch"** gesetzt.

   2. Befindet sich der Befehl außerhalb einer Kondition, wird die Abfolge abgebrochen.

4. **Der letzte gelesene Block bleibt im Buffer** erhalten. Diese ist bereits dem Parameter entsprechend manipuliert, jedoch ist **keine "*{text}*" Trennung** enthalten.
5. Ist der „Read Block" Befehl innerhalb einer Kondition, wird die Zeichenkette im Buffer gespeichert und der **Success-Flag** auf **„Wahr"** gesetzt.

2.9.4.3.4 Flussdiagramm



2.9.4.4 Blockweise Schreiben

Beschreibt und speichert einen oder mehrere Blöcke auf einen Tag, wobei die Art der **Eingabe** verschieden sein kann.

2.9.4.4.1 Syntax

Eingabe als Byte	**Eingabe als Nibble**	**Eingabe als Dezimal** (16 Bit)
W[]	W{}	
W[*a*]	W{*a*}	
W[*a-b*]	W{*a-b*}	
		W(*a-b*,*w*,*x*)
W[*a-b*,"*{text}*"]	W{*a-b*,"*{text}*"}	W(*a-b*,"*{text}*")
W[*a-b*,*c*,"*{text}*"]	W{*a-b*,*c*,"*{text}*"}	W(*a-b*,*c*,*d*,"*{text}*")

2.9.4.4.2 Parametererklärung

Parameter	**Erklärung**
leer	Schreiben des 0. Blocks
a	Schreiben des **a.** Blocks
a-b	Schreiben des **a. bis b.** Blocks
c	**c.** Byte als **erste Position** verwenden (Bei Dezimal: High-Byte) **wenn nicht angegeben: erstes Bytes** verwenden (Bei Dezimal: Low-Byte)
d	Nur bei Dezimal-Eingabe: **d.** Byte als **zweite Position** verwenden; (Low-Byte) **wenn nicht angegeben: nur** w. Byte als Position verweden (LowByte)
"*{text}*"	Die Zeichenkette **{text}** enthält die zu beschreibende Informationen. Bei *Byte-Eingabe* [ ] : Es gelten die gleichen Regeln wie bei dem **Text-Befehl -** siehe Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.** Max. 4 Bytes (Blocklänge eines Tags) Bei *Nibble-Eingabe*{ } : Die Zeichen 'A' bis 'F' müssen groß geschrieben werden. Max. 8 Bytes bzw. die Länge muss eine gerade Zahl sein. Bei *Dezimal-Eingabe* ( ) : Die Zahl darf max. 65535 betragen. Hexadezimalen werden mit einem 'H' vorher gekennzeichnet: max. HFFFF **Wenn nicht angegeben:** Wenn der Speicher den gewünschten Block bereits eingelesen hat, werden die Informationen vom Speicher gelesen – nur bei Byte- und Nibble-Eingabe. Nur sinnvoll zusammen mit **„Release Tag"**.

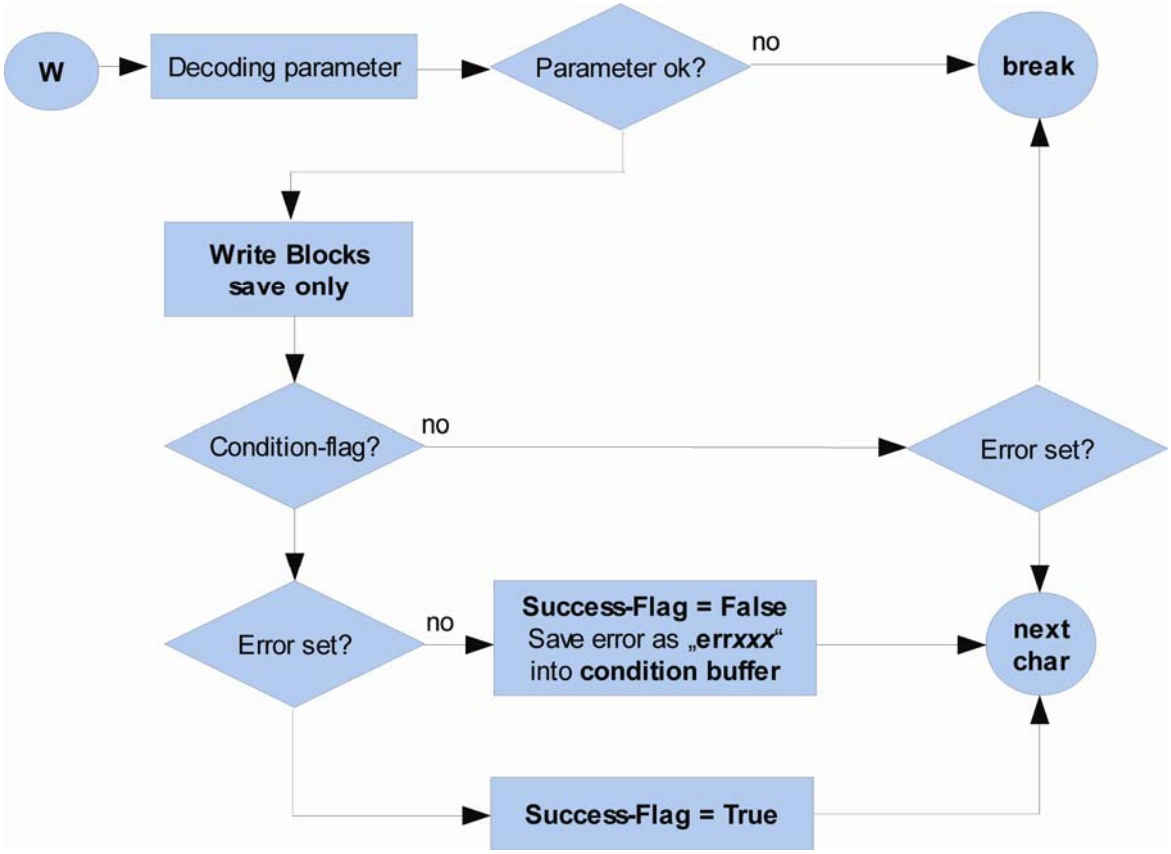
2.9.4.4.3 Erklärung

1. Parameter wird auf Syntax und Gültigkeit überprüft und gespeichert.
2. Tritt ein Kommunikationsfehler auf (z.B. Fehlerüberschreitung) gibt es 2 Möglichkeiten:

   1. Befindet sich der Befehl innerhalb einer Kondition, wird der Fehlercode in Form von „err*xxx*", wobei *xxx* gleich dem Fehlercode ist, als Zeichenkette gespeichert. Der **Success-Flag** wird auf **„Falsch"** gesetzt.

   2. Befindet sich der Befehl außerhalb einer Kondition, wird die Abfolge abgebrochen.
3. Ist der „Write Block" Befehl innerhalb einer Kondition, wird der **Success-Flag** auf **„Wahr"** gesetzt.

2.9.4.4.4 Flussdiagramm



2.9.4.5 Byte weise Lesen

Ermöglicht das Byte weise Auslesen des Speicherinhalts eines Tags. Der maximale Speicher errechnet sich aus der Blocklänge in Byte und aus der Anzahl der auslesbaren bzw. beschreibbaren Blöcke. Diese Parameter sind mit dem Befehl „%mb set tag“ einstellbar – siehe Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.**. Hierbei sollte beachtet werden, dass das eigentliche Einlesen des Tags dennoch Block weise erfolgt und dementsprechend auch gespeichert wird. Jedoch wird die Ausgabe Block-unabhängig behandelt, was den Vorteil hat, einen **selbst definierten Speicheraufbau** erzeugen zu können.

2.9.4.5.1 Syntax

Ausgabe in Byte	**Ausgabe in Nibble**	**Ausgabe in Dezimal** (16 Bit)
PR[]	PR{}	PR()
PR[*a*]	PR{*a*}	PR(*a*)
PR[*a*-*b*]	PR{*a*-*b*}	PR(*a*-*b*)

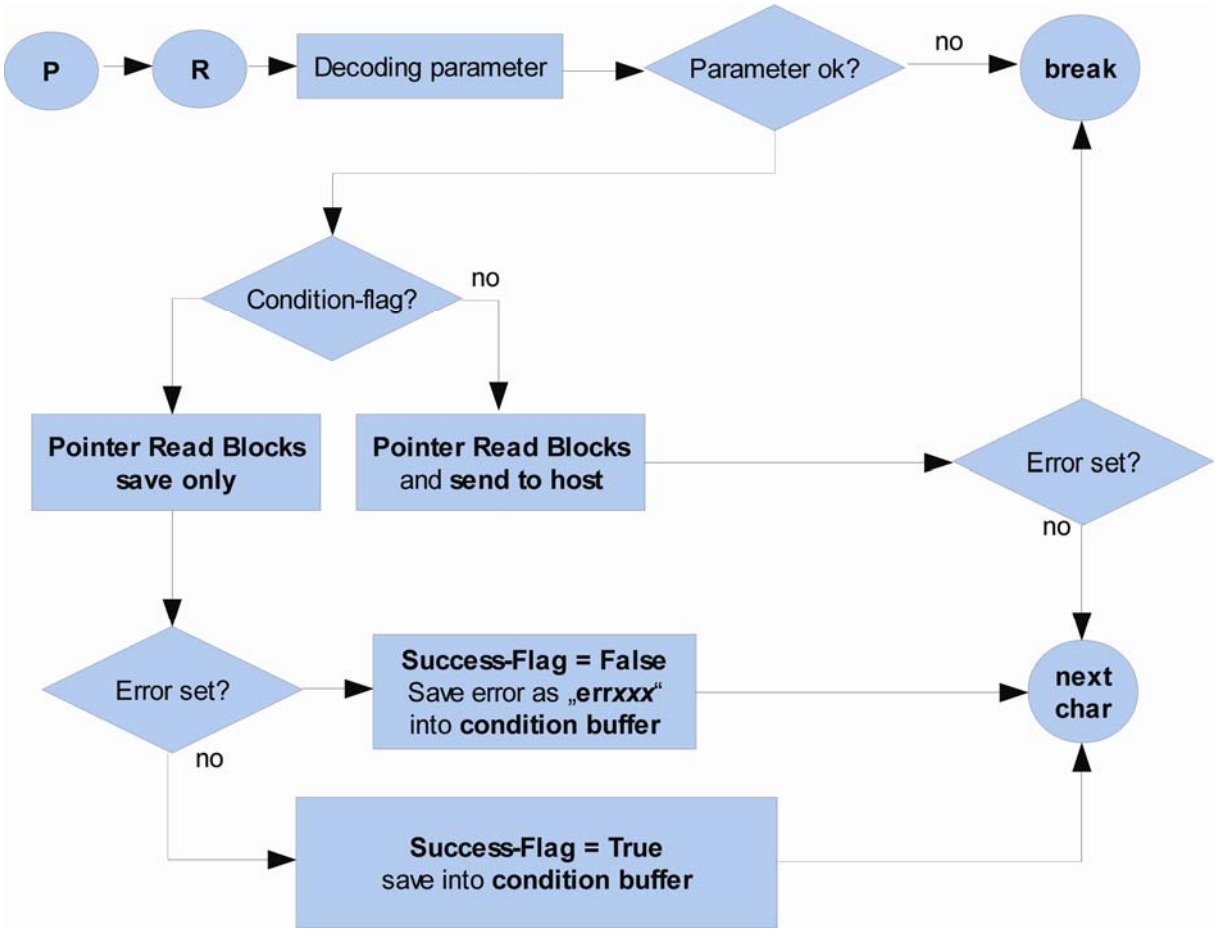
2.9.4.5.2 Parametererklärung

Parameter	**Erklärung**
leer	Alles auslesen (siehe „%mb get tag“ Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.**)
a	Lesen des **a.** Bytes (bei Dezimal das **a.** Byte als **Low-Byte** behandeln)
a-b	Lesen des **a. bis b.** Bytes (bei Dezimal das **a.** Byte als **Low-Byte** und das **b.** Byte als **High-Byte** behandeln)

2.9.4.5.3 Erklärung

1. Parameter wird auf Syntax und Gültigkeit überprüft und gespeichert.
2. Ist der Befehl **innerhalb einer Kondition** wird die Funktion **'Pointer Read' ohne Ausgabe** aufgerufen. Ist er **außerhalb einer Kondition** wird die Funktion **mit Ausgabe** aufgerufen.
3. Tritt ein Kommunikationsfehler auf (z.B. Fehlerüberschreitung) gibt es 2 Möglichkeiten:
   1. Befindet sich der Befehl innerhalb einer Kondition, wird der Fehlercode in Form von „err*xxx*“, wobei *xxx* gleich dem Fehlercode ist, als Zeichenkette gespeichert. Der **Success-Flag** wird auf **„Falsch“** gesetzt.
   2. Befindet sich der Befehl außerhalb einer Kondition, wird die Abfolge abgebrochen.
4. Ist der „Pointer Read“ Befehl innerhalb einer Kondition, wird die Zeichenkette im Buffer gespeichert und der **Success-Flag** auf **„Wahr“** gesetzt.

2.9.4.5.4 Flussdiagramm



2.9.4.6 Byte weise Schreiben

Ermöglicht das Byte weise Beschreiben des Speicherinhalts eines Tags. Der maximale Speicher errechnet sich aus der Blocklänge in Byte und aus der Anzahl der auslesbaren bzw. beschreibbaren Blöcke. Diese Parameter sind mit dem Befehl „%mb set tag“ einstellbar – siehe Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.**.
Hierbei sollte beachtet werden, dass das eigentliche Beschreiben des Tags dennoch Block weise erfolgt und dementsprechend auch gespeichert wird. Jedoch wird die Eingabe Block-unabhängig behandelt, was den Vorteil hat, einen **selbst definierten Speicheraufbau** erzeugen zu können.

2.9.4.6.1 Syntax

Eingabe in Byte	**Eingabe in Nibble**	**Eingabe in Dezimal** (16 Bit)
PW[]	PW{}	
PW[“*{text}*“]	PW{“*{text}*“}	PW(“*{text}*“)
PW[*a*,“*{text}*“]	PW{*a*,“*{text}*“}	PW(*a*,“*{text}*“)

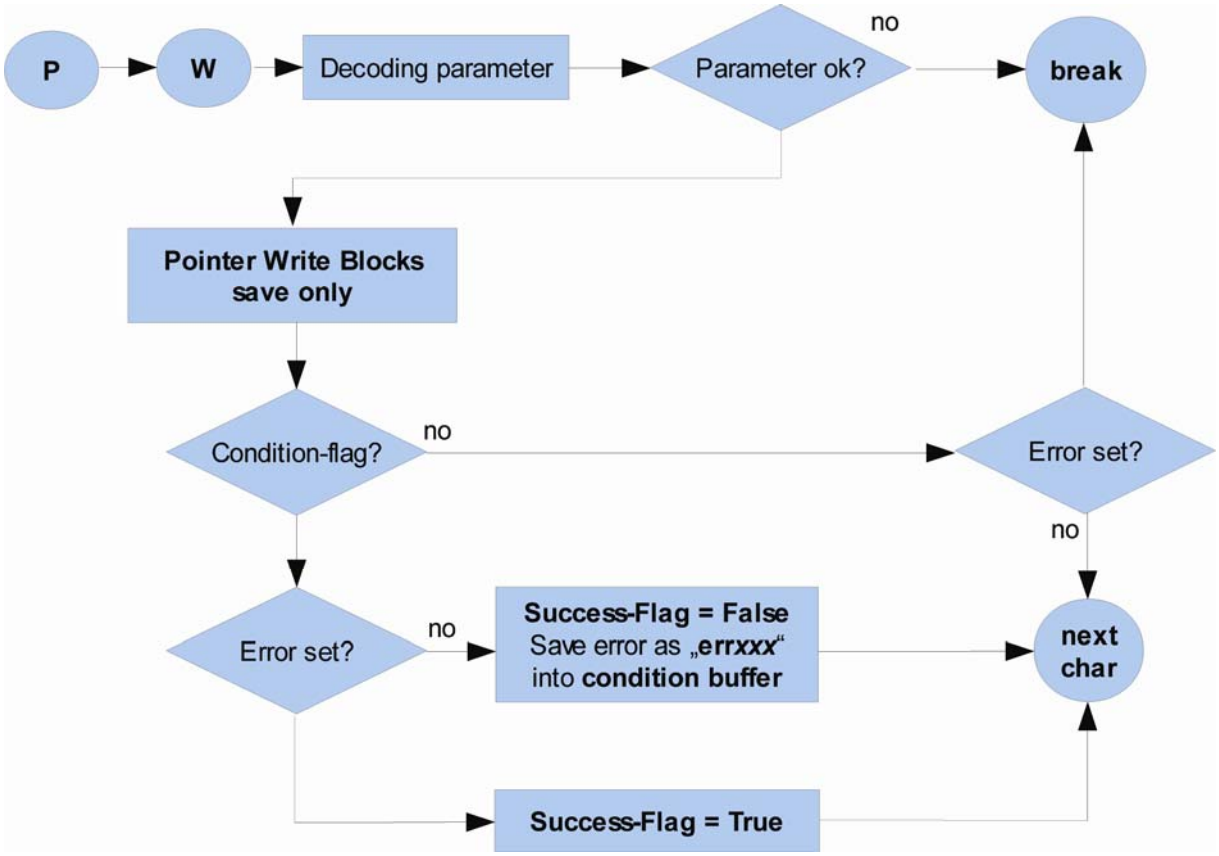
2.9.4.6.2 Parametererklärung

Parameter	**Erklärung**
leer	Alles mit 0 beschreiben (siehe „%mb get tag“ Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.**)
a	Schreiben ab dem **a.** Byte (wenn nicht angegeben → ab dem ersten Byte)
“*{text}*“	Die Zeichenkette **{text}** enthält die zu beschreibende Informationen. Aus der Länge der Zeichenkette errechnet sich die Anzahl der zu beschreibenden Bytes. Bei *Byte-Eingabe* [ ] : Es gelten die gleichen Regeln wie bei dem **Text-Befehl -** siehe Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.** Bei *Nibble-Eingabe*{ } : Die Zeichen 'A' bis 'F' müssen groß geschrieben werden. Max. 8 Bytes bzw. die Länge muss eine gerade Zahl sein. Bei *Dezimal-Eingabe* ( ) : Die Zahl darf max. 65535 betragen. Hexadezimalen werden mit einem 'H' vorher gekennzeichnet: max. HFFFF

2.9.4.6.3 Erklärung

1. Parameter wird auf Syntax und Gültigkeit überprüft und gespeichert.
2. Tritt ein Kommunikationsfehler auf (z.B. Fehlerüberschreitung) gibt es 2 Möglichkeiten:
    1. Befindet sich der Befehl innerhalb einer Kondition, wird der Fehlercode in Form von „err*xxx*“, wobei *xxx* gleich dem Fehlercode ist, als Zeichenkette gespeichert. Der **Success-Flag** wird auf **„Falsch“** gesetzt.
    2. Befindet sich der Befehl außerhalb einer Kondition, wird die Abfolge abgebrochen.
3. Ist der „Pointer Write“ Befehl innerhalb einer Kondition, wird der Success-Flag auf „Wahr“ gesetzt.

2.9.4.6.4 Flussdiagramm



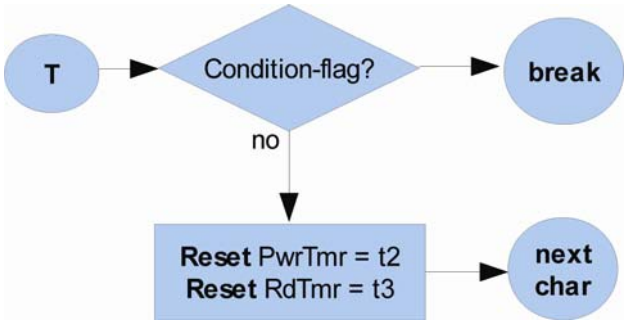
Trigger Timeout

Wenn MultiBlock automatisiert angewandt wird, werden die Zeitzähler im Vollbetrieb (siehe Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.**) nicht mehr selbstständig zurückgesetzt. Hier kann der Anwender bestimmen, wann oder unter welcher Bedingung eine Rücksetzung der Zeitzähler erfolgen soll.

2.9.4.6.5 Syntax

T

2.9.4.6.6 Flussdiagramm



2.9.4.7 Tag Release

Um ein mehrmaliges Auslesen oder Beschreiben eines Tags – z.B. im automatisierten Betrieb – zu vermeiden, ist der Einsatz des Befehls „Tag Release“ zu empfehlen. Dieser Befehl kann in einer Bedingung verwendet werden und ermöglicht dem Anwender während einer Abfolge selbst die Konsequenz festzulegen.

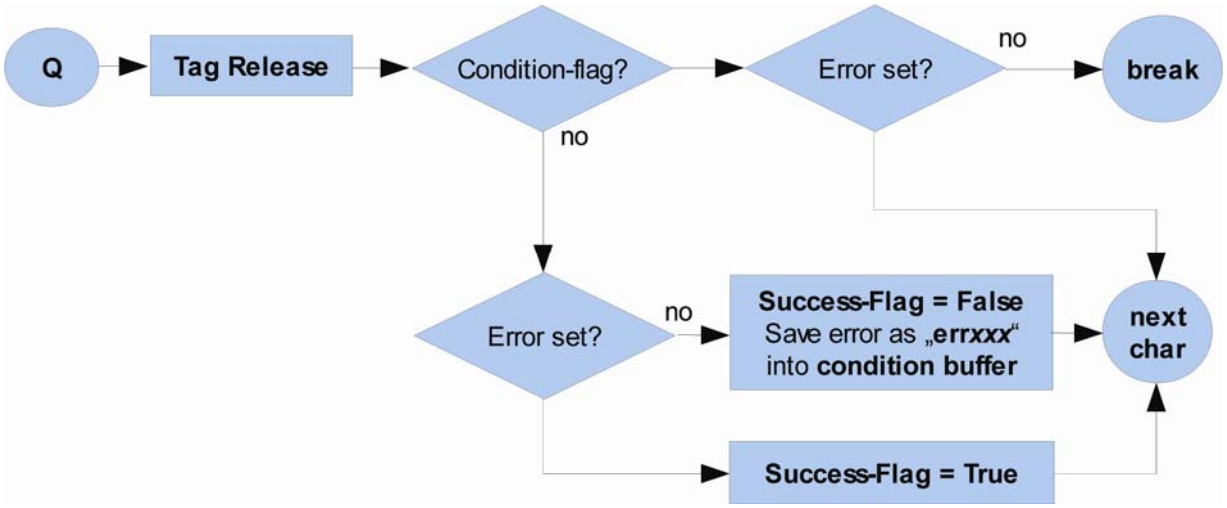
2.9.4.7.1 Syntax

Q

2.9.4.7.2 Erklärung

1. Die Funktion „Tag Release“ setzt das HF-Modul in „continuous mode“, und wartet bis **ein anderer Tag** (UID) im Feld, oder **der Tag selbst nicht mehr** im Feld ist.
2. Tritt ein Kommunikationsfehler auf (z.B. Zeitüberschreitung von 3 Sekunden) gibt es zwei Möglichkeiten:
    1. Befindet sich der Befehl innerhalb einer Kondition, wird der Fehlercode in Form von „err*xxx*“, wobei *xxx* gleich dem Fehlercode ist, als Zeichenkette gespeichert. Der **Success-Flag** wird auf **„Falsch“** gesetzt.
    2. Befindet sich der Befehl außerhalb einer Kondition, wird die Abfolge abgebrochen.
3. Ist der „Tag Release“ Befehl innerhalb einer Kondition, wird der **Success-Flag** auf **„Wahr“** gesetzt.

2.9.4.7.3 Flussdiagramm



2.9.4.8 Beep

Der Anwender hat die Möglichkeit, Tonsignale erzeugen zu lassen – z.B. nach einem erfolgreichen Lese- oder Schreibvorgang. Dabei ist die Höhe, Dauer und Lautstärke des Signals frei wählbar.

2.9.4.8.1 Syntax

b[]
b[*a*]
b[*a*,*b*]
b[*a*,*b*,*c*]
b[*a*,*b*,*c*,*d*]

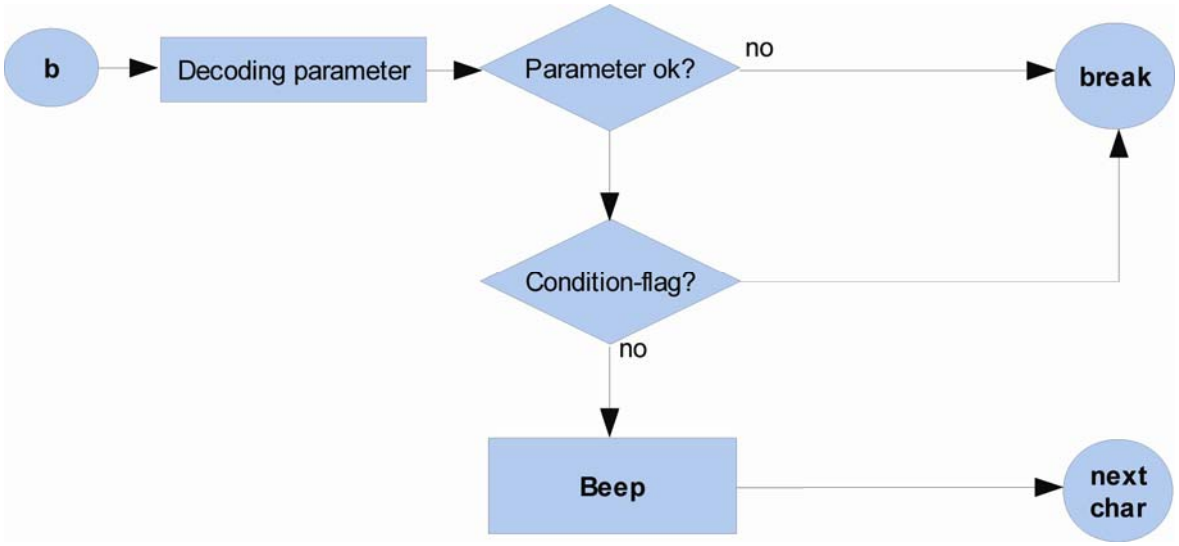
2.9.4.8.2 Parametererklärung

Parameter	**Erklärung**
leer	Frequenz ist 300Hz; Dauer ist 100ms; Lautstärke ist 3; Fortsetzung erfolgt sofort
a	**a** enspricht der Frequenz f.ton = **a***10Hz (z.B.: 44 entspricht 440Hz) *Dauer ist 100ms; Lautstärke ist 3; Fortsetzung erfolgt sofort*
b	**b** enspricht der Dauer t.ton = **b***100ms (z.B.: 10 entspricht 1 Sekunde) *Lautstärke ist 3; Fortsetzung erfolgt sofort*
c	**c** enspricht der Lautstärke 0 ist Minimum und 3 ist Maximum *Fortsetzung erfolgt sofort*
d	**d** kann 0 oder 1 sein. Bei 0 wird die Abfolge sofort fortgesetzt und bei 1 wird die Dauer des Signals abgewartet. **Vor einem Lese- oder Schreibvorgang** wird das **Abwarten** des Signalendes, also **d=1**, empfohlen.

2.9.4.8.3 Erklärung

1. Parameter wird auf Syntax und Gültigkeit überprüft und gespeichert.
2. Das „Beep“ Befehl wird entsprechend dem Parameter ausgeführt.

2.9.4.8.4 Flussdiagramm



2.9.4.9 Imager Decode – Befehl

Falls ein Imager-Modul installiert ist, kann man innerhalb der Abfolge auch einen Decode Befehl senden und dann die empfangene Zeichenkette ausgeben oder vergleichen.

2.9.4.9.1 Syntax

Ausgabe in Byte
I[]
I[*a*]
I[*a*-*b*]

2.9.4.9.2 Parametererklärung

Parameter	**Erklärung**
leer	Alles auslesen.
a	Lesen des **a.** Bytes
a-b	Lesen des **a. bis b.** Bytes

2.9.4.9.3 Erklärung

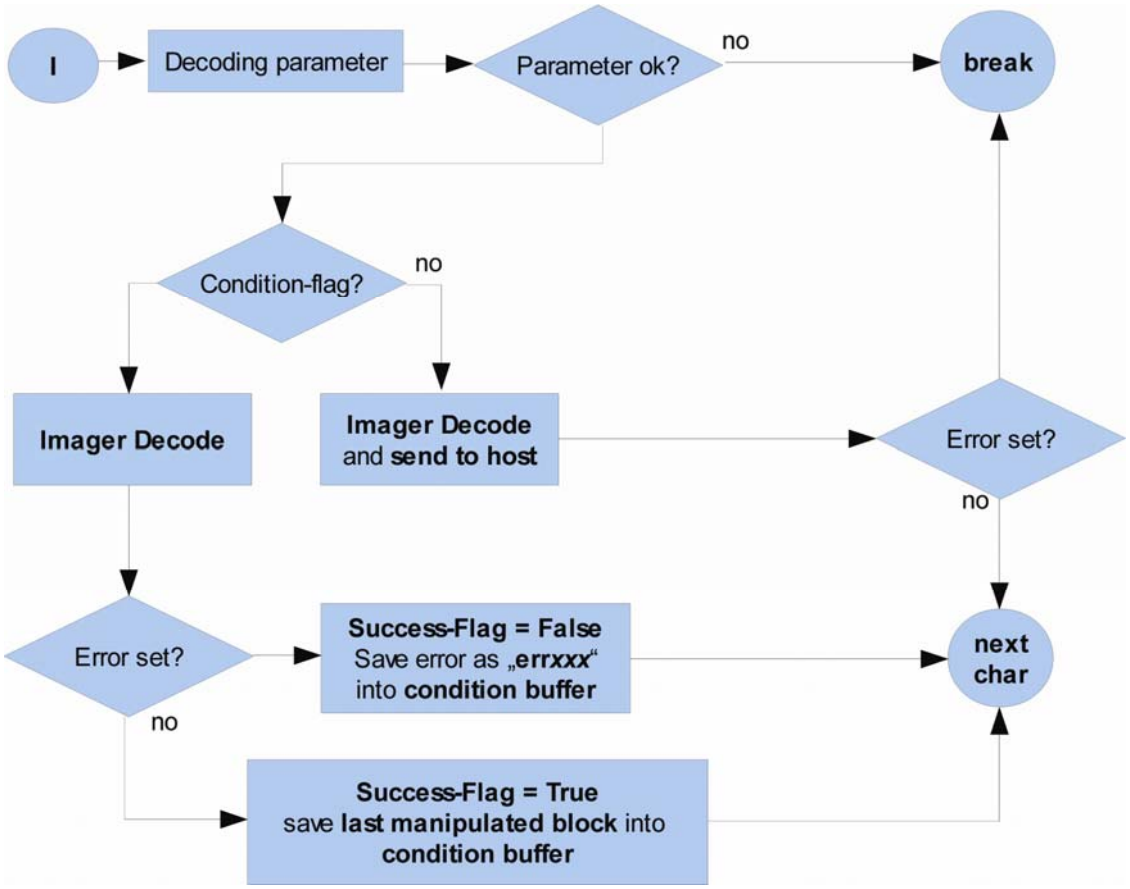
1. Parameter wird auf Syntax und Gültigkeit überprüft und gespeichert.
2. Ist der Befehl **innerhalb einer Kondition** wird die Funktion **'Imager Decode' ohne Ausgabe** aufgerufen. Ist er **außerhalb einer Kondition** wird die Funktion **mit Ausgabe** aufgerufen.
3. Tritt ein Kommunikationsfehler auf (z.B. Zeitüberschreitung) gibt es 2 Möglichkeiten:

   1. Befindet sich der Befehl innerhalb einer Kondition, wird der Fehlercode in Form von „err*xxx*“, wobei *xxx* gleich dem Fehlercode ist, als Zeichenkette gespeichert. Der **Success-Flag** wird auf **„Falsch“** gesetzt.

   2. Befindet sich der Befehl außerhalb einer Kondition, wird die Abfolge abgebrochen.

4. Ist der „Imager Decode“ Befehl innerhalb einer Kondition, wird die Zeichenkette im Buffer gespeichert und der **Success-Flag** auf **„Wahr“** gesetzt.

2.9.4.9.4 Flussdiagramm



2.9.4.10 Konditionen

Oft ist es nötig Speicherinhalte oder UIDs zu überprüfen. Mit vier Zeichen ist dies mit MultiBlock möglich:

?	If	
:	then	Ergebnis = true
/	else	Ergebnis = false
;	end if	

2.9.4.10.1 ? .. wenn

Sobald das **„wenn"-Zeichen ?** erkannt wird, wird der **„Condition-Flag"** gesetzt und signalisiert allen anderen nachkommenden Befehlen, dass diese sich in einer Kondition (Bedingung) befinden. Zwei Eigenschaften sind in einer Bedingung bei jedem Befehl gleich:

1. Es erfolgt **keine Ausgabe** an den Host.
2. **Kein Abbruch** der Abfolge durch Kommunikationsfehler bzw. Timeouts.

Die Tabelle im Kapitel 2.9.4 zeigt unter anderem die Befehle, die innerhalb einer Kondition verwendet werden können. Bis auf den Befehl „Release Tag" können grundsätzlich alle Befehle, mit Hilfe der Operatoren, mit Zahlen oder Zeichenketten verglichen werden.

Mögliche Operatoren innerhalb einer Kondition sind:

Zeichen	Bedeutung	Anwendung bei
<	kleiner	Zahlen
<=	kleiner gleich	Zahlen
>	größer	Zahlen
>=	größer gleich	Zahlen
=	gleich	Zahlen & Zeichenketten
!=	ungleich	Zahlen & Zeichenketten

Manche Befehle können auch ohne Operatoren angewandt werden – diese setzen den **„Success-Flag"**, der den Erfolg z.B. einer Kommunikation mit „Wahr" (=erfolgreich) oder „Falsch" (=nicht erfolgreich) festlegt.

Es ist nur ein Vergleich in einer Bedingung möglich.

2.9.4.10.2 : .. Dann

Das **„:" - Zeichen** bildet den **Abschluss einer Bedingung** und setzt somit den „Condition-Flag" zurück. Wurde ein Operator angewendet und sind zwei Variablen angegeben worden, werden diese auf ihre Gültigkeit überprüft und verglichen. Das Ergebnis ist dann „Wahr" oder „Falsch".

Wurde kein Operator angewandt, wird überprüft ob der „Success-Flag" gesetzt wurde und diese dann weitergegeben.

Falls das Ergebnis **„Falsch"** ist, wird **zum nächsten** „/" - **Zeichen** (=ansonsten) **gesprungen**.

Falls das Ergebnis **„Wahr"** ist, wird normal **das nächste Zeichen** ausgewertet.

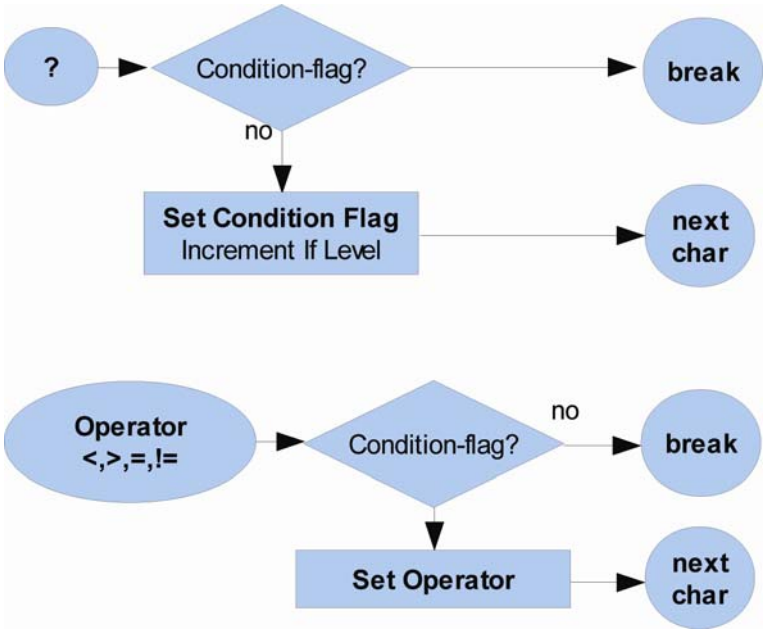
2.9.4.10.3 / .. Ansonsten

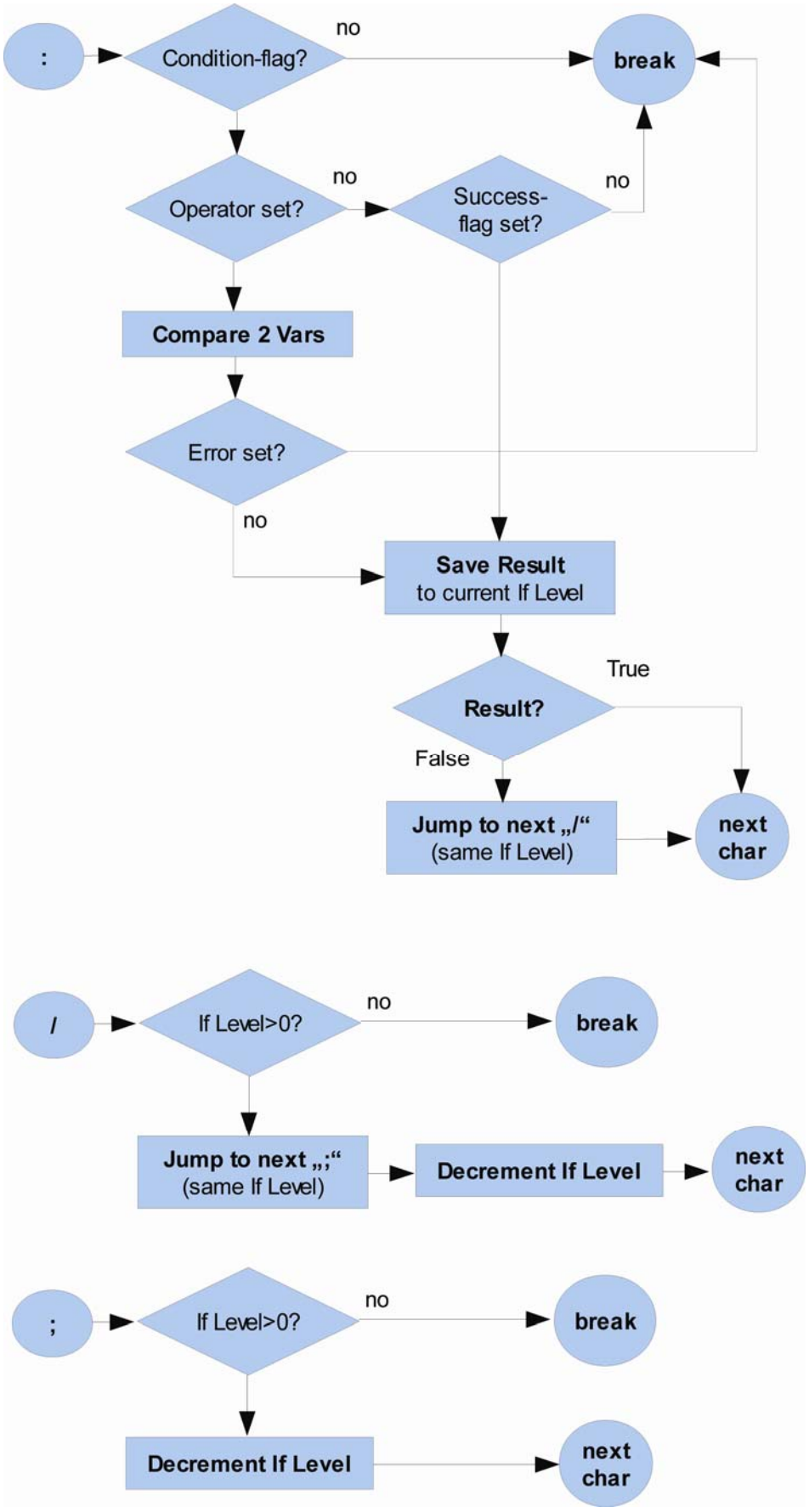
Das **„/" - Zeichen** ist nur für ein „Wahr"-Ergebnis relevant, und signalisiert, dass bis zum „;"-Zeichen (Wenn Ende) gesprungen werden soll. Sind keine Befehle für ein „Falsch"-Ergebnis vorgesehen, so muss das „/"-Zeichen trotzdem den Abschluss für diese Befehlskette bilden.

2.9.4.10.4 ; .. Wenn Ende

Dieses Zeichen bildet **das Ende einer „Wenn"-Anweisung**.

2.9.4.10.5 Flussdiagramm




:
Condition-flag?
no
break
Operator set?
no
Success-
flag set?
no
Compare 2 Vars
Error set?
no
Save Result
to current If Level
Result?
True
False
Jump to next „/“
(same If Level)
next
char
/
If Level>0?
no
break
Jump to next „;“
(same If Level)
Decrement If Level
next
char
;
If Level>0?
no
break
Decrement If Level
next
char

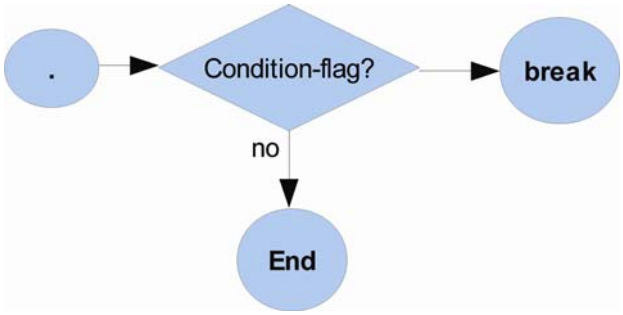
2.9.4.11 End – Befehl

Der End-Befehl beendet die Abfolge und kann überall, außer innerhalb einer Kondition, verwendet werden.

2.9.4.11.1 Syntax

.

2.9.4.11.2 .Flussdiagramm



2.9.5 Fehlercodes

In diesem Kapitel werden alle möglichen Fehlercodes innerhalb einer Abfolge aufgelistet.

Fehlercode	**Beschreibung**
50	unknown command
51	no value
52	value expected
53	value out of range
54	unexpected end
55	if - condition error
56	if - level overflow
57	if - level is zero
58	if - level - other failure
59	if - condition expected
60	if - condition - operator already set
61	if - condition - can not compare
62	if - condition - wrong operator
63	if - condition - can not exec command
64	hex value expected
65	if - condition - unknow operator
66	text expected
67	converting text failed
68	parameter - fast already set
69	parameter - brace open expected
70	parameter - wrong brace type
71	parameter - too many parameters
72	parameter - unknown parameter
73	parameter - wrong output type
74	parameter - need more parameters
75	parameter - text-parameter already set
76	block length - out of range
77	converting nibble string failed
78	converting nibble to byte failed
79	converting byte to nibble failed
100	microengine error - X
101	microengine error - F
102	microengine error - C
103	microengine error - N
104	microengine error - Q
105	microengine error - U

106	microengine error - R
107	microengine error - O
108	wrong mode
109	timeout reset
110	timeout get uid
111	timeout select
112	timeout read
113	timeout write
114	timeout stop continuous mode
115	write failed
116	pointer read wrong parameter
117	pointer read overflow
118	pointer write wrong parameter
119	pointer write overflow
120	wrong uid
121	release tag command - failed
150	beep command - wrong parameter
160	imager command - failed

Technischer Support

Online Support

You can find the latest builds, updates, workaround for problems and Frequently Asked Question (FAQ) under the support section on the DATATRONIC Web site. If you can not solve the problem on your own, please contact our office listed in the topic Contact Information. http://www.datatronic.eu/support

Contact Information

Worldwide, Technical Support is available through our office:

DATATRONIC IDentsysteme GmbH
Dreisteinstraße 47
AT 2372 Gießhübl
AUSTRIA

Tel.: 0043 (0) 2236 / 377668-0
Fax: 0043 (0) 2236 / 377668-11

mail@datatronic.eu
http://www.datatronic.eu

DATATRONIC IDentsysteme GmbH

Dreisteinstraße 47
AT 2372 Gießhübl
AUSTRIA

Tel.: 0043 (0) 2236 / 377668-0
Fax: 0043 (0) 2236 / 377668-11

mail@datatronic.eu
http://www.datatronic.eu

TagTrans ProgrammersGuide Rev. 1.0

CUSTOMER Order # xxx xxx - 001
Manufacturer Part # xxx xxx - 001