Programmer's Guide

C-Control IoT-Starter Kit 10

Gültig ab:

- **Firmware Version: 01v000**
- **Server Version: 43.3**
- **Hardware Version: 1.1**

300822 | Rev.01

Microtronics
We live M2M

Kapitel 1 Inhaltsverzeichnis

Kapitel 2 Allgemeine Angaben

Die Informationen dieses Handbuchs wurden sorgfältig geprüft und nach bestem Wissen zusammengestellt. Der Hersteller übernimmt dennoch keine Verantwortung für möglicherweise in diesem Handbuch enthaltene falsche Angaben. Der Hersteller ist nicht verantwortlich für direkte, indirekte, versehentliche oder Folgeschäden, die aus Fehlern oder Unterlassungen in diesem Handbuch entstanden, selbst wenn auf die Möglichkeit solcher Schäden hingewiesen wurde. Im Interesse der fortlaufenden Produktentwicklung behält sich der Hersteller jederzeit und ohne vorherige Ankündigung oder Verpflichtung das Recht auf Verbesserungen an diesem Handbuch und der hierin beschriebenen Produkte vor.

> ***Hinweis:*** *Die Angaben dieses Handbuches sind ab den auf der Titelseite angeführten Versionsständen gültig. Überarbeitete Ausgaben dieses Handbuchs sowie Software und Treiber-Updates sind im Servicebereich des C-Control-Servers erhältlich.*

2.1 Übersetzung

Bei Lieferungen in die Länder des europäischen Wirtschaftsraumes ist das Handbuch in die Sprache des Verwenderlandes zu übersetzen. Sollten im übersetzten Text Unstimmigkeiten auftreten, ist das Original-Handbuch (deutsch) zur Klärung heranzuziehen oder der Hersteller zu kontaktieren.

2.2 Copyright

2.3 Gebrauchsnamen

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen und dgl. in diesem Handbuch berechtigen nicht zu der Annahme, dass solche Namen ohne weiteres von jedermann benutzt werden dürfen; oft handelt es sich um gesetzlich geschützte eingetragene Warenzeichen, auch wenn sie nicht als solche gekennzeichnet sind.

Kapitel 3 rapidM2M Toolset

Hinweis: *Alle Screenshots zeigen das rapidM2M Toolset in der Version 1.18. Bei neueren Versionen können geringfügige Änderungen am Erscheinungsbild des Programms vorgenommen worden sein.*

3.1 Allgemein

Das rapidM2M Toolset steht unter folgender Adresse gratis zum Download bereit:

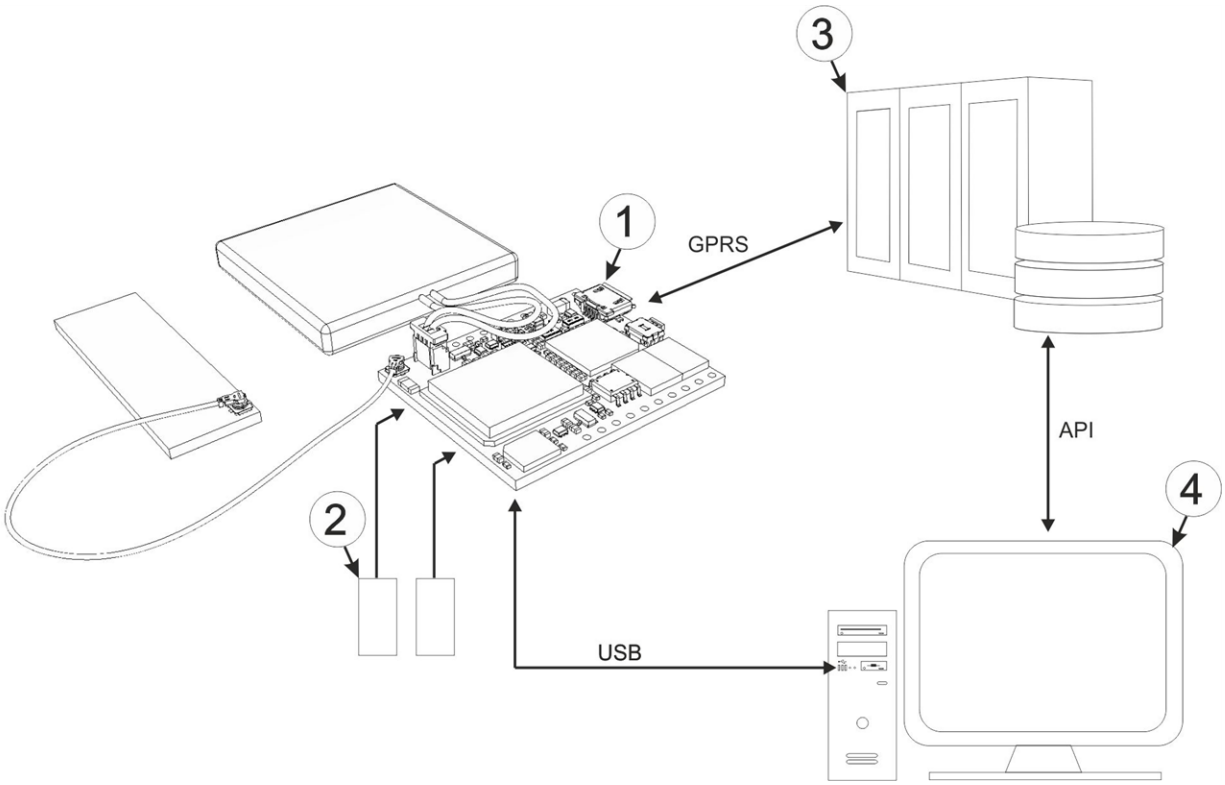
www.microtronics.at/Toolset

Es handelt sich um eine IDE, die den Kunden bei der Erstellung von Applikationen für das C-Control IoT-Starter Kit 10 unterstützen soll. Diese umfasst den kompletten Entwicklungsprozess - angefangen vom Editieren des PAWN-Scripts (siehe "Pawn Script" auf Seite 25), über das Kompilieren des Scripts in Byte-Code und Laden des Byte-Codes in das C-Control IoT-Starter Kit 10 bis hin zum Auslesen der vom C-Control IoT-Starter Kit 10 an den C-Control-Server übertragenen Daten über die API des C-Control-Servers. Das rapidM2M Toolset ermöglicht auch die Anzeige der Debug-Informationen, die innerhalb des PAWN-Scripts mittels der Funktionen "print" und "printf" auf die Standardausgabe gedruckt wurden und das Laden des Log-Files aus dem C-Control IoT-Starter Kit 10 . Die 10 Konfigurationsspeicherblöcke können ebenfalls mittels rapidM2M Toolset aus dem C-Control IoT-Starter Kit 10 gelesen bzw. in das C-Control IoT-Starter Kit 10 geschrieben werden. Das rapidM2M Toolset kann auch dazu verwendet werden, Firmwareupdates in das C-Control IoT-Starter Kit 10 einzuspielen und enthält eine Sammlung von Beispiel-Scripts.

3.2 Voraussetzungen

Schnittstellen	1 x USB
Betriebssystem	Win XP Windows Vista Windows 7
Internetverbindung	empfohlen
Benötigter Speicherplatz	ca. 75MB

3.3 Funktionsprinzip



Funktionsprinzip

1	C-Control IoT-Starter Kit 10	3	C-Control-Sever
2	Sensor, der mit dem C-Control IoT-Starter Kit 10 verbunden ist (optional)	4	PC, auf dem das rapidM2M Toolset installiert ist

Das rapidM2M Toolset kommuniziert über die USB-Schnittstelle direkt mit dem C-Control IoT-Starter Kit 10 . Dazu gehören:

- der Script-Download (siehe "Kateireiter "Script"" auf Seite 17)
- der Up/Download der Konfigurationsspeicherblöcke (siehe "Karteireiter "Config"" auf Seite 19)
- die Anzeige der Konsolenausgaben (siehe "Karteireiter "Console"" auf Seite 20)
- das Auslesen der Log-Einträge (siehe "Karteireiter "Log"" auf Seite 22)
- das Firmwareupdate (siehe "Karteireiter "FW"" auf Seite 23)

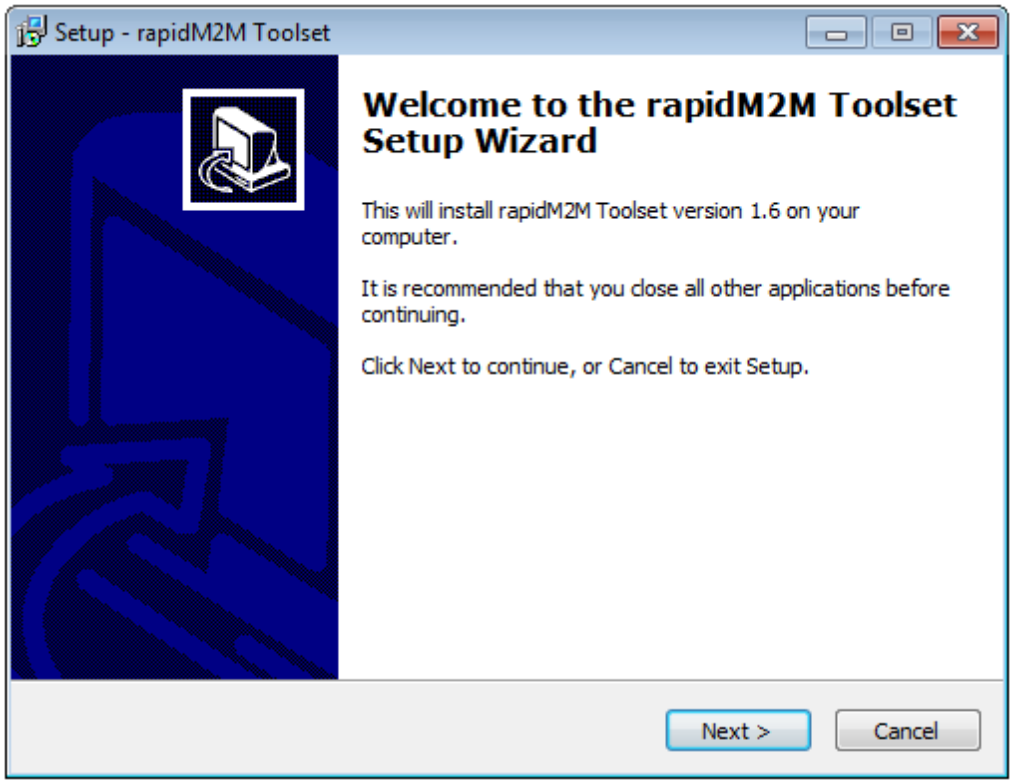
Das rapidM2M Toolset bietet auch die Möglichkeit die Daten über die API des C-Control-Severs auszulesen (siehe "Karteireiter "Server"" auf Seite 21). Dadurch kann sehr einfach und schnell überprüft werden, ob die komplette Prozesskette - angefangen beim Speichern der Daten mittels Script über die Übertragung per GPRS bis hin zur Ablage der Daten am Server - korrekt funktioniert.

3.4 Installation

Das folgende Kapitel beschreibt den Installationsprozess unter Windows 7.

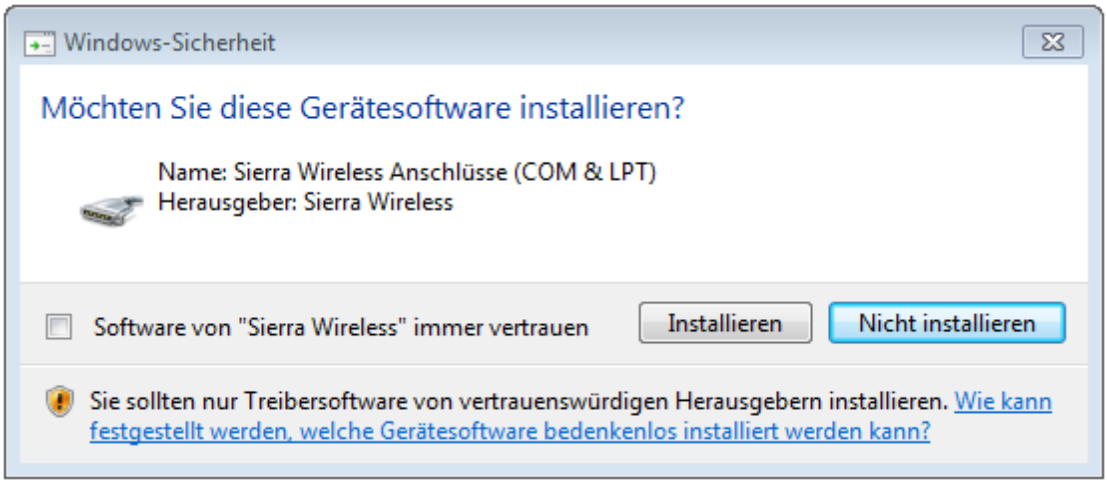
1. Führen Sie die Datei *"InstRapidM2MToolset.exe"* aus, um den Installationsprozess zu starten.

   Hinweis: *Verbinden Sie das C-Control IoT-Starter Kit 10 erst nach Abschluss des Installationsprozesses mit Ihrem PC da die benötigten USB-Treiber erst während dieses Vorgangs installiert werden.*

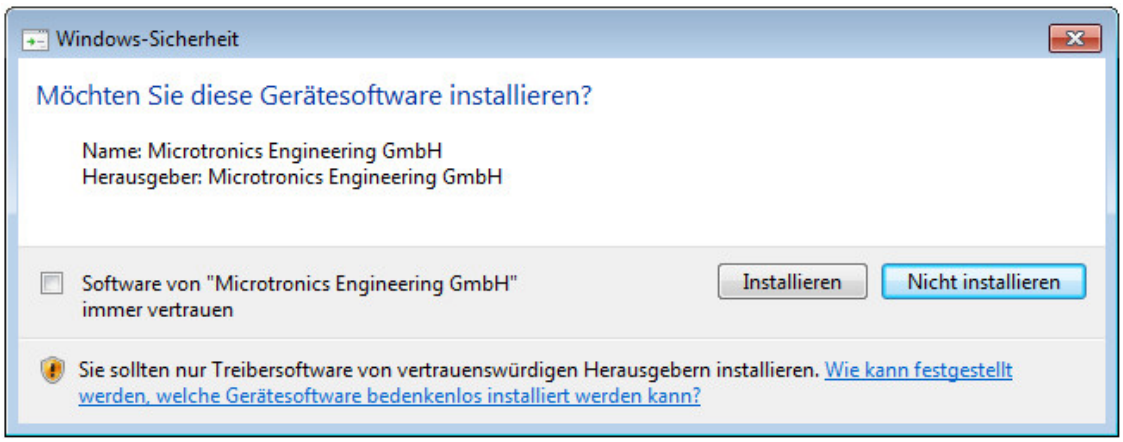


rapidM2M Toolset Setup Wizard

2. Folgen Sie den Anweisungen des Setup Wizzards bis Sie zu der folgenden Ansicht gelangen. Für den ordnungsgemäßen Betrieb müssen die USB-Treiber zwingend installiert werden.

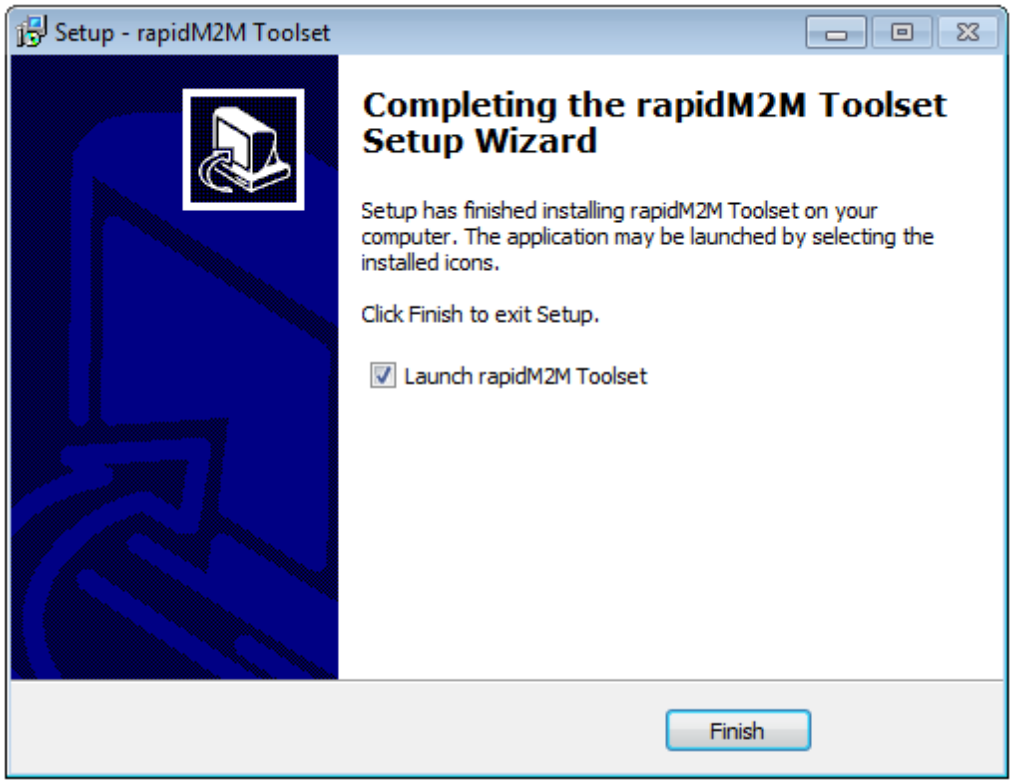


Installation der USB-Treiber für M12x Module



Installation der USB-Treiber für M22x Module und C-Control IoT-Starter Kit 10

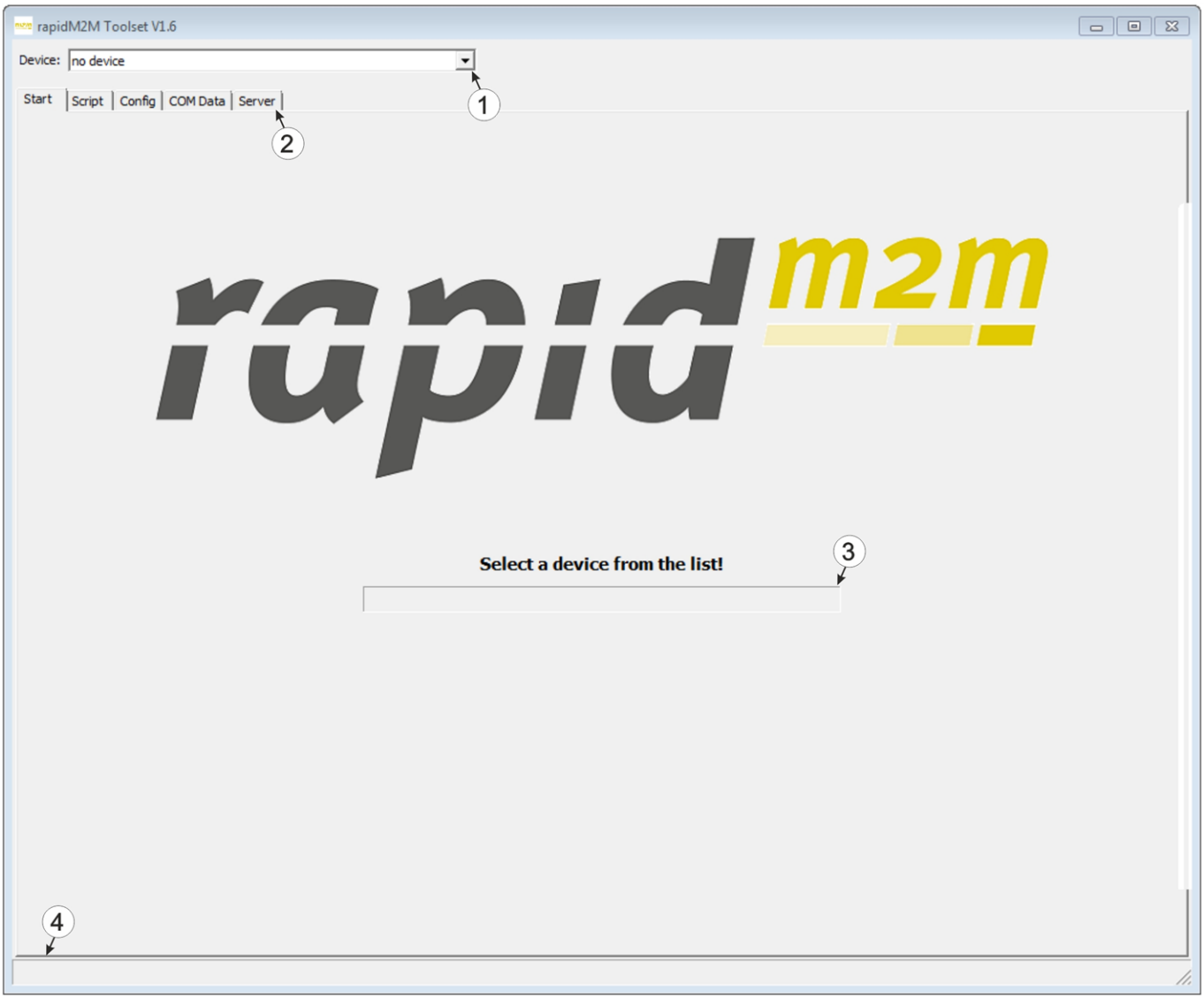
3. Wenn Sie schließlich zur folgenden Ansicht gelangen, schließen Sie den Installationsvorgang durch Klicken auf den Button *"Finish"* ab.



Setup abschließen

4. Nachdem Sie auf den Button *"Finish"* geklickt haben, startet die Installation der USB Treiber.

3.5 Übersicht



Karteireiter "Start" (kein C-Control IoT-Starter Kit 10 ausgewählt)

1	Auswahl des C-Control IoT-Starter Kit 10	3	Fortschrittsbalken für den Aufbau der USB-Verbindung zum C-Control IoT-Starter Kit 10
2	Auswahl der Karteireiter für die verschiedenen Funktionen des rapidM2M Toolset , die auch ohne verbundenem C-Control IoT-Starter Kit 10 verfügbar sind	4	Anzeige des Kommunikationsstatus auf der USB-Schnittstelle zum C-Control IoT-Starter Kit 10

Sobald die USB-Verbindung zu einem C-Control IoT-Starter Kit 10 hergestellt wurde, werden die Karteireiter der Funktionen, die eine bestehende USB-Verbindung zum C-Control IoT-Starter Kit 10 erfordern, eingeblendet und der Karteireiter "Start" ausgeblendet. Zusätzlich werden im rechten oberen Bereich des rapidM2M Toolset weitere Informationen zum C-Control IoT-Starter Kit 10 eingeblendet.

Wichtiger Hinweis: *Verknüpfen Sie das C-Control IoT-Starter Kit 10 mit einer Messstelle (siehe "Benutzerhandbuch für C-Control -Server " 206.886) bevor Sie per USB darauf zugreifen. Bei einer nachträglichen Verknüpfung würden alle bisher getroffenen Einstellungen (inkl. Script) durch die Übernahme der Einstellungen von der Messstelle verloren gehen. Die Meldung "`Warning: No Context available! All settings and script will be lost when device is assigned to a site`" in der Anzeige für den Kommunikationsstatus auf der USB-Schnittstelle weist darauf hin, dass noch keine Messstelle mit dem C-Control IoT-Starter Kit 10 verknüpft ist.*

Verbindung zum C-Control IoT-Starter Kit 10 hergestellt

1	Auswahl des C-Control IoT-Starter Kit 10	4	Anzeige der GSM-Signalstärke
2	Auswahl der Karteireiter für die verschiedenen Funktionen des rapidM2M Toolset	5	Seriennummer des verbundenen C-Control IoT-Starter Kit 10
3	Anzeige des GSM-Verbindungsstatus. Durch Klicken auf dieses Symbol kann ein Verbindungsaufbau gestartet werden.		

GSM-Verbindungsstatus

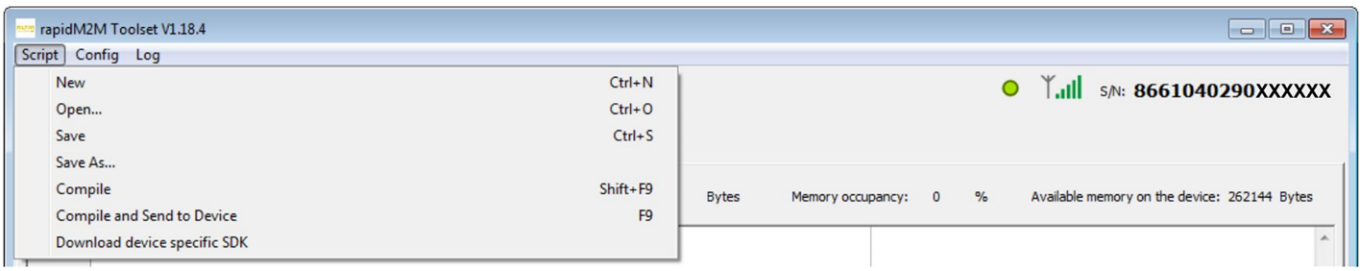
Statussymbol	
	offline
	offline, Wakeup über den C-Control-Server möglich
, blinkend	Verbindung wird hergestellt
	verbunden
	letzter Verbindungsversuch ist fehlgeschlagen
	letzter Verbindungsversuch ist fehlgeschlagen, Wakeup über den C-Control-Server möglich

GSM-Signalstärke

	› -64dBm
	-64...-73dBm
	-74...-83dBm
	-84...-93dBm
	-94...-107dBm
	<= -108dBm

3.6 Menü des rapidM2M Toolset

3.6.1 Script

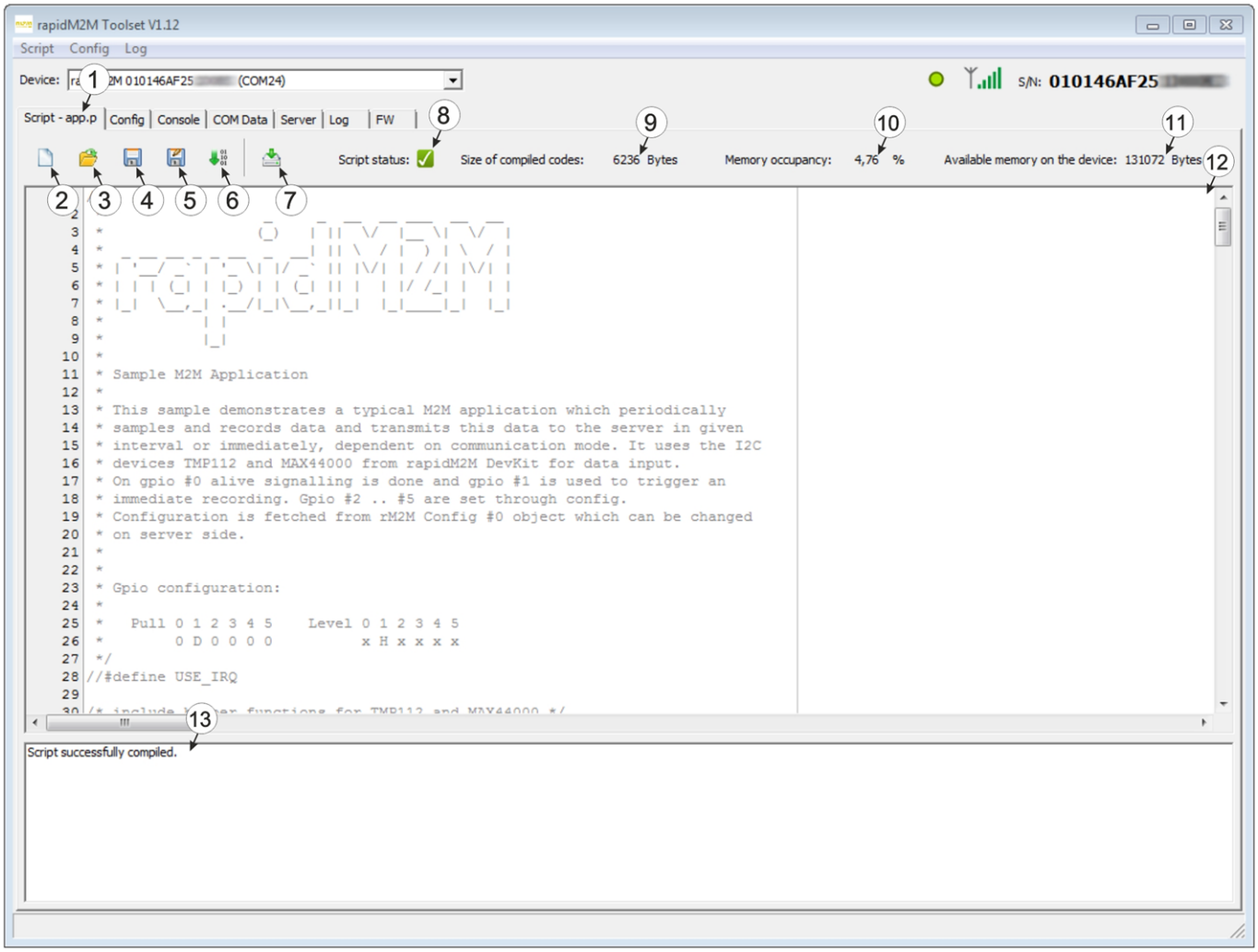


Menüpunkt "Script"

Über den Menüpunkt "Script -› Download device specific SDK" lassen sich gerätespezifische Erweiterungen für das rapidM2M Toolset herunterladen und installieren. Dazu muss das entsprechende Gerät mit der USB-Schnittstelle des PCs verbunden sein und der PC über eine bestehende Verbindung zum Internet verfügen.

3.7 Kateireiter "Script"

Dieser Bereich dient zum Erstellen des Scripts. Der Karteireiter ist auch verfügbar, wenn keine USB-Verbindung zu einem C-Control IoT-Starter Kit 10 besteht. Mit Ausnahme des Downloads des kompilierten Scripts und der Anzeige der Speichergrößen ("Size of compiled codes", "Memory occupancy" und "Available memory on the device"), stehen in diesem Fall alle Funktionen zur Verfügung. Das zuletzt geöffnete Script wird beim Programmstart automatisch geladen. Sollte das gerade geöffnete Script durch ein anderes Programm verändert worden sein, erfolgt eine Meldung mit der Möglichkeit das geänderte Script in den Script-Editor zu laden.



Karteireiter "Script"

1	Dateiname des geöffneten Scripts. Wurde das Script geändert jedoch noch nicht gespeichert, wird dies durch "*" nach dem Dateinamen signalisiert.
2	erstellt ein neues leeres Script
3	öffnet ein gespeichertes Script
4	speichert das aktuell geöffnete Script

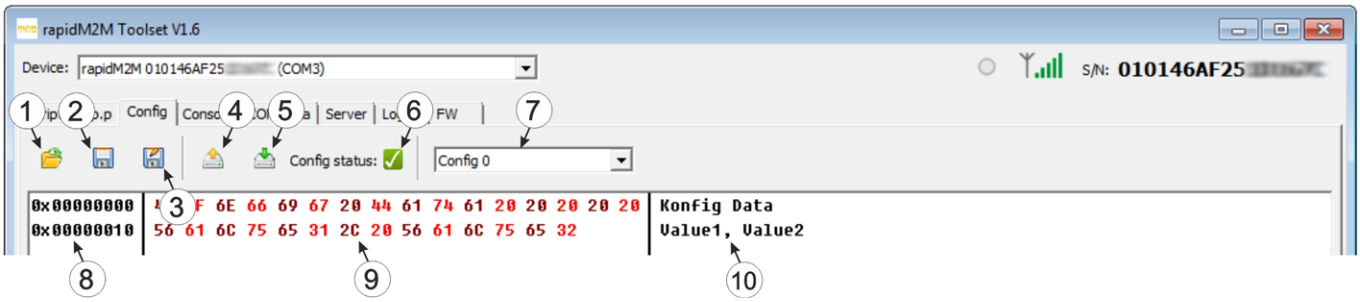
5	öffnet den Dialog zum Speichern des Scripts unter einem anderen Dateinamen
6	kompiliert das geöffnete Script. Dabei wird es auch automatisch gespeichert.
7	überträgt das geöffnete Script in das Modul/Gerät. Dabei wird das Script zuerst gespeichert und dann kompiliert. Das kompilierte PAWN-Binary (*.amx) wird im selben Ordner wie das Script gespeichert.
8	Status der Script-Übertragung (siehe "Script Status")
9	Größe des kompilierten Codes
10	gibt die Belegung des verfügbaren Speichers in % an
11	gibt die Größe des verfügbaren Speichers am C-Control IoT-Starter Kit 10 an (Angabe der komprimierten Größe)
12	Script-Editor
13	Debugausgabe des Script-Kompilers. Durch einen Doppelklick auf einen Eintrag in der Debugausgabe springt der Curser im Script-Editor zur betroffenen Zeile.

Script Status

Symbol	Beschreibung
	Script wurde erfolgreich in das C-Control IoT-Starter Kit 10 geladen.
	Script wird gerade zum C-Control IoT-Starter Kit 10 übertragen.
	Der Zeitstempel des im C-Control IoT-Starter Kit 10 gespeicherten kompilierten Scripts stimmt nicht mit dem des kompilierten Scripts des rapidM2M Toolset überein.

3.8 Karteireiter "Config"

Dieser Karteireiter ermöglicht den Zugriff auf die 10 Konfigurationsspeicherblöcke. Diese können editiert, gespeichert und vom und zum C-Control IoT-Starter Kit 10 übertragen werden. Der Karteireiter ist auch verfügbar, wenn keine USB-Verbindung zu einem C-Control IoT-Starter Kit 10 besteht. In diesem Fall ist es allerdings nicht möglich, die Konfigurationsblöcke vom und zum C-Control IoT-Starter Kit 10 zu übertragen.



Karteireiter "Config"

1	öffnet einen gespeicherten Konfigurationsspeicherblock	6	Status der Konfigurationsblockübertragung (siehe "Config Status")
2	speichert den aktuell geöffneten Konfigurationsspeicherblock	7	Auswahl des Konfigurationsspeicherblocks[1)]
3	öffnet den Dialog zum Speichern des aktuell geöffneten Konfigurationsspeicherblocks unter einem anderen Dateinamen	8	Adressspalte
4	lädt den ausgewählten Konfigurationsspeicherblock vom C-Control IoT-Starter Kit 10	9	Hex-Ansicht des Konfigurationsspeicherblocks
5	überträgt den ausgewählten Konfigurationsspeicherblock ans C-Control IoT-Starter Kit 10	10	ASCII-Ansicht des Konfigurationsspeicherblocks

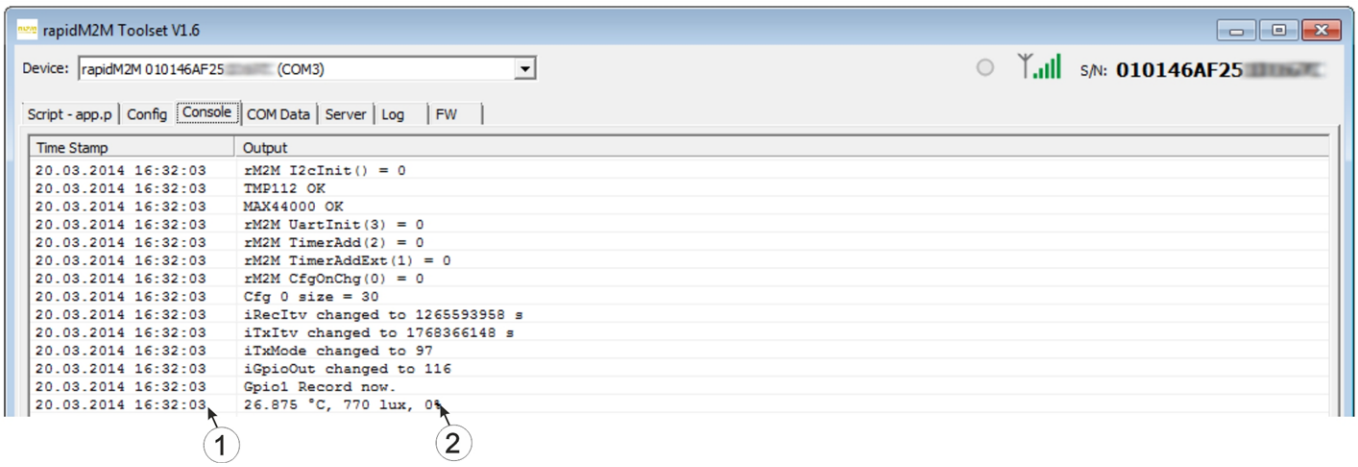
[1)] Die Operationen "öffnen", "speichern" und "übertragen" beziehen sich immer nur auf den ausgewählten Konfigurationsspeicherblock.

Config Status

Symbol	Beschreibung
✓	Der ausgewählte Konfigurationsspeicherblock wurde erfolgreich in das C-Control IoT-Starter Kit 10 geladen.
⚙	Der ausgewählte Konfigurationsspeicherblock wird gerade übertragen.
!	Der Zeitstempel des im C-Control IoT-Starter Kit 10 gespeicherten Konfigurationsspeicherblock stimmt nicht mit dem des rapidM2M Toolset überein. Die Anzeige gilt für den aktuell ausgewählten Konfigurationsspeicherblock.

3.9 Karteireiter "Console"

Dieser Bereich dient der Anzeige von Debug-Informationen, die innerhalb des PAWN-Scripts mittels der Funktionen "print" und "printf" (siehe "Consolen Funktionen" auf Seite 29) auf die Standardausgabe gedruckt wurden. Der Karteireiter "Console" ist nur verfügbar, wenn eine USB-Verbindung zu einem C-Control IoT-Starter Kit 10 besteht.



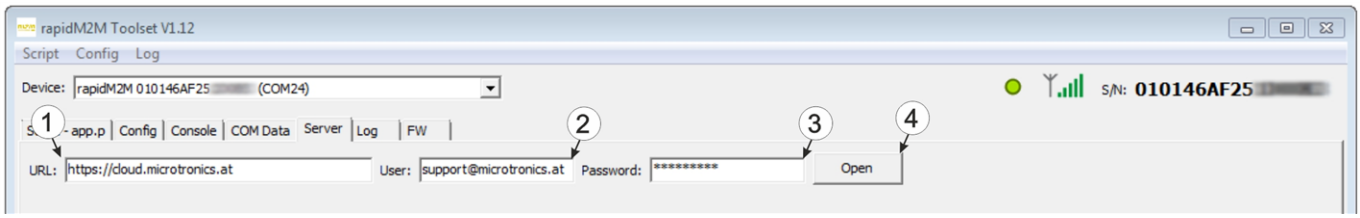
Karteireiter "Console"

1 Zeitpunkt, zu dem die Debug-Informationen über die USB-Schnittstelle empfangen wurden	**2** Anzeige der Debug-Informationen

Durch einen Klick mit der rechten Maustaste auf einen der Einträge der Debug-Anzeige wird ein Kontextmenü eingeblendet, dass das Löschen sowie das Speichern der Liste ermöglicht.

3.10 Karteireiter "Server"

Über diesen Karteireiter erreichen Sie die Testseite der REST-API des C-Control-Servers. Mit ihrer Hilfe können Sie sehr einfach und schnell überprüften, ob die komplette Prozesskette - angefangen beim Speichern der Daten mittels Script, über die Übertragung per GPRS, bis hin zur Ablage der Daten am Server - korrekt funktioniert. Diese Funktion kann auch ohne über die USB-Schnittstelle verbundenes C-Control IoT-Starter Kit 10 verwendet werden. In diesem Fall muss zusätzlich zu Serveradresse, Benutzername und Passwort auch noch die Seriennummer des Geräts eingegeben werden (siehe "Karteireiter "Server" ohne verbundenem C-Control IoT-Starter Kit 10 " auf Seite 21).



Karteireiter "Server" mit verbundenem C-Control IoT-Starter Kit 10

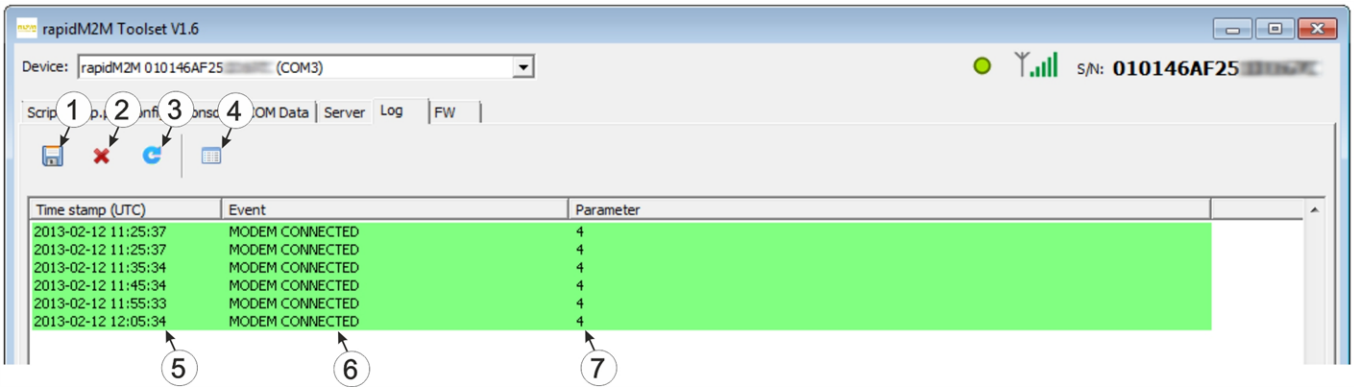
1	Adresse des Servers (C-Control-Server)	3	Passwort für das Benutzerkonto
2	Benutzername eines am C-Control-Server angelegten Benutzerkontos	4	öffnet die Testseite der REST-API des C-Control-Servers im Browser

Karteireiter "Server" ohne verbundenem C-Control IoT-Starter Kit 10

1	Seriennummer des C-Control IoT-Starter Kit 10 auf dessen Daten per REST-API zugegriffen werden sollen

3.11 Karteireiter "Log"

Dieser Bereich dient der Verwaltung der Log-Einträge. Er ermöglicht das Laden der Einträge vom C-Control IoT-Starter Kit 10 , das Speichern als *.tsv-Datei und das Löschen der Einträge aus dem Speicher des C-Control IoT-Starter Kit 10 .

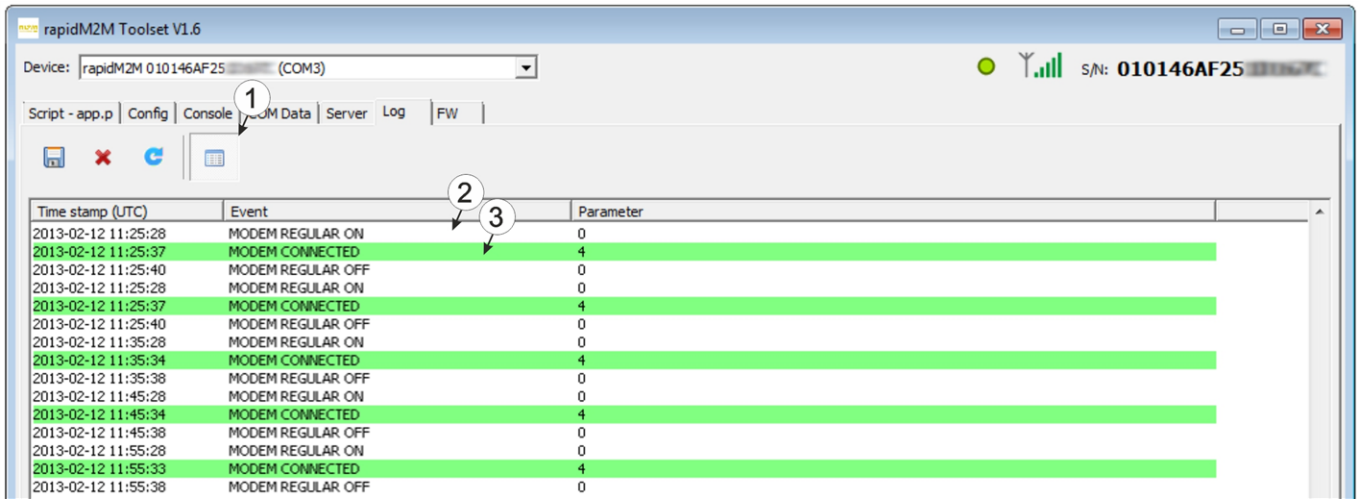


Karteireiter "Log"

1	speichert die angezeigten Log-Einträge als *.tsv-Datei	5	Zeitstempel des Log-Eintrags
2	löscht die Log-Einträge aus dem Speicher des C-Control IoT-Starter Kit 10	6	Log-Eintrag
3	lädt die Log-Einträge vom C-Control IoT-Starter Kit 10	7	Parameter des Log-Eintrags
4	aktiviert die detaillierte Darstellung der Log-Einträge		

Die farbliche Markierung gibt Aufschluss darüber, wie kritisch der Log-Eintrag zu bewerten ist. Die weiß gekennzeichneten informativen Log-Einträge werden nur angezeigt, wenn die detaillierte Darstellung der Log-Einträge aktiviert ist (siehe "Karteireiter "Log" mit aktivierter Detailansicht" auf Seite 23).

Farbe	Bewertung
weiß	Information über den aktuellen Betriebszustand
grün	
hellblau	
blau	
lila	
grau	
gelb	unkritischer Fehler
rot	kritischer Fehler

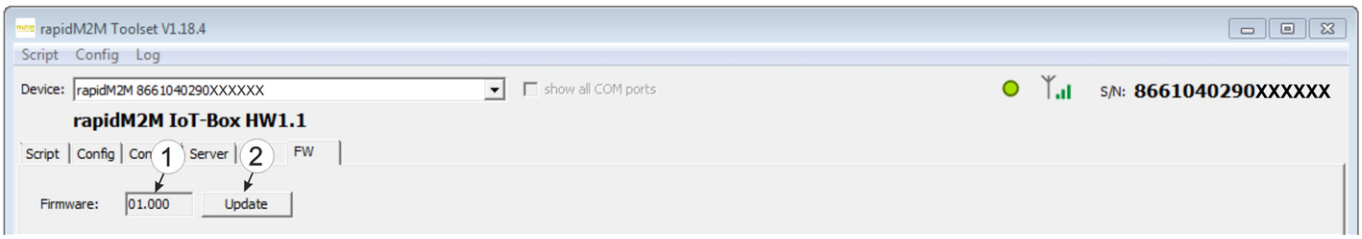


Karteireiter "Log" mit aktivierter Detailansicht

1 detaillierte Darstellung der Log-Einträge aktiviert	3 Log-Eintrag, der in jedem Fall angezeigt wird
2 informativer Log-Eintrag, der nur sichtbar ist, wenn die detaillierte Darstellung aktiviert wurde	

3.12 Karteireiter "FW"

Dieser Bereich ermöglicht das direkte Einspielen der Firmware über die USB-Schnittstelle.



Karteireiter "FW"

1 aktuell installierte Firmware	2 öffnet den Dialog zur Auswahl der Firmware-Datei

3.13 Hot-Keys

Befehl	Tastenkombination	Erklärung
New Script	Ctrl+N	erstellt ein neues leeres Script
Open Script	Ctrl+O	öffnet ein gespeichertes Script
Save Script	Ctrl+S	speichert das aktuell geöffnete Script
Compile Script	Shift+F9	kompiliert das geöffnete Script. Dabei wird es auch automatisch gespeichert.
Compile and Send to Device	F9	überträgt das geöffnete Script in das Modul/Gerät. Dabei wird das Script zuerst gespeichert und dann kompiliert. Das kompilierte PAWN-Binary (*.amx) wird im selben Ordner wie das Script gespeichert.

Kapitel 4 Pawn Script

4.1 Allgemein

Das folgende Kapitel beschreibt die Funktionalität des Pawn Scripts. PAWN (vormals SMALL) ist eine C-ähnliche Skriptsprache, welche auf embedded Systemen läuft.

Zusätzliche detaillierte Informationen finden Sie auf der Website der Entwickler: http://www.compuphase.com/pawn/pawn.htm.

Es gibt mehrere Möglichkeiten, um ein Pawn Script für das C-Control IoT-Starter Kit 10 zu erstellen:

- Direkte Eingabe in das Eingabefenster „Script“ im Konfigurationsabschnitt „Steuerung“
- Hochladen eines zuvor erstellten Binary-Files (*.amx) auf den C-Control–Server
- Benutzen des Script-Editors des rapidM2M Toolset

4.1.1 Direkte Eingabe eines Pawn Scripts

Die Eingabe des Pawn Scripts erfolgt über den Konfigurationsabschnitt „Steuerung“ (siehe Benutzerhandbuch für C-Control -Server " 206.886) der Eingabemaske zur Konfiguration der Messstelle. Als „Script Type“ muss „Pawn“ ausgewählt werden, damit das C-Control IoT-Starter Kit 10 die unter „Script“ eingegebenen Befehle als Pawn Script interpretiert.

4.1.2 Hochladen eines Binary-Files

Wurde über die Listenauswahl "Script Quelle" im Konfigurationsabschnitt „Steuerung“ (siehe Benutzerhandbuch für C-Control -Server " 206.886) der Eingabemaske zur Konfiguration der Messstelle der Eintrag "Hochladen eines kompilierten Scripts" ausgewählt, kann ein zuvor mittels z.B. rapidM2M Toolset (siehe "Kateireiter "Script"" auf Seite 17) erstelltes Binary-File auf den C-Control–Server hochgeladen werden. Dieses wird dann bei der nächsten Verbindung in das C-Control IoT-Starter Kit 10 geladen. Als „Script Type“ muss auch bei dieser Methode „Pawn“ ausgewählt werden, damit das C-Control IoT-Starter Kit 10 die Befehle als Pawn Script interpretiert.

4.1.3 Script-Editor des rapidM2M Toolset

Das rapidM2M Toolset beinhaltet unter dem Kateireiter "Script" (siehe "Kateireiter "Script"" auf Seite 17) einen Editor zum Erstellen und Kompilieren der Pawn Scripts. Zum Funktionsumfang des rapidM2M Toolset gehört auch das Übertragen der kompilierten Scripts per USB-Verbindung in das C-Control IoT-Starter Kit 10 .

4.2 rapidM2M API

4.2.1 Core Funktionen

native heapspace();

liefert den freien Speicherplatz auf dem Heap

	Erklärung
Rückgabewert	*Der freie Speicherplatz auf dem Heap. Der Stack und der Heap besetzen einen gemeinsamen Speicherbereich, so dass dieser Wert die Anzahl der Bytes angibt, die entweder für den Stack oder den Heap übrig sind.*

native funcidx(const name[]);

liefert den Index einer öffentlichen Funktion

Parameter	*Erklärung*
name	*Name der öffentlichen Funktion*

	Erklärung
Rückgabewert	• *-1, wenn keine Funktion mit dem übergebenen Namen existiert* • *Index der öffentlichen Funktion*

native numargs();

liefert die Anzahl der an eine Funktion übergebenen Argumente

	Erklärung
Rückgabewert	*Die Anzahl der Argumente, die an eine Funktion übergeben wurden. numargs ist nützlich innerhalb von Funktionen mit einer variablen Argumentenliste.*

native getarg(arg, index=0);

liefert den Wert des Arguments

Parameter	*Erklärung*
arg	*Die Sequenznummer des Arguments. Verwenden Sie 0 für das erste Argument.*
index	*Index, falls sich "arg" auf ein Array bezieht*

	Erklärung
Rückgabewert	*Diese Funktion liefert ein Argument aus einer variablen Argumentenliste. Wenn das Argument ein Array ist, gibt "index" den Index des gewünschten Arrayelements an. Der Rückgabewert ist der Wert des Arguments.*

native setarg(arg, index=0, value);

setzt den Wert des Arguments

Parameter	*Erklärung*
arg	*Die Sequenznummer des Arguments. Verwenden Sie 0 für das erste Argument.*
index	*Index, falls sich "arg" auf ein Array bezieht*
value	*Wert auf den das Argument gesetzt werden soll*

	Erklärung
Rückgabewert	• *true, wenn der Wert gesetzt werden konnte* • *false, wenn das Argument oder der Index ungültig sind* *Diese Funktion setzt ein Argument in einer variablen Argumentenliste. Wenn das Argument ein Array ist, gibt "index" den Index des gewünschten Arrayelements an.*

native tolower(c);

wandelt ein Zeichen in einen Kleinbuchstaben um

Parameter	*Erklärung*
c	*Zeichen, das in einen Kleinbuchstaben umgewandelt werden soll*

	Erklärung
Rückgabewert	*Die Kleinbuchstaben-Variante des übergebenen Zeichens, falls vorhanden, oder der unveränderte Zeichencode von "c", wenn der Buchstabe "c" kein Kleinbuchstaben-Äquivalent hat.*

native toupper(c);

wandelt ein Zeichen in einen Großbuchstaben um

Parameter	*Erklärung*
c	*Zeichen, das in einen Großbuchstaben umgewandelt werden soll*

	Erklärung
Rückgabewert	*Die Großbuchstaben-Variante des übergebenen Zeichens, falls vorhanden, oder der unveränderte Zeichencode von "c", wenn der Buchstabe "c" kein Großbuchstaben-Äquivalent hat.*

native swapchars(c);

vertauscht die Reihenfolge der Bytes

Parameter	*Erklärung*
c	*Wert für den die Bytes vertauscht werden sollen*

	Erklärung
Rückgabewert	*Wert, bei dem die Bytes im Parameter "c" vertauscht sind (das niedrigste Byte wird das höchste Byte)*

native min(value1, value2);

liefert den kleineren der beiden übergebenen Wert

Parameter	*Erklärung*
value1	*zwei Werte, von denen der kleinere ermittelt werden soll*
value2	

	Erklärung
Rückgabewert	*der kleinere der beiden übergebenen Werte*

native max(value1, value2);

liefert den größeren der beiden übergebenen Wert

Parameter	*Erklärung*
value1	*zwei Werte, von denen der größere ermittelt werden soll*
value1	

	Erklärung
Rückgabewert	*der größere der beiden übergebenen Werte*

native clamp(value, min=cellmin, max=cellmax);

prüft, ob der übergebene Wert zwischen "min" und "max" liegt

Parameter	*Erklärung*
value	*Wert, der geprüft werden soll*
min	*untere Bereichsgrenze*
max	*obere Bereichsgrenze*

	Erklärung
Rückgabewert	• *"value", wenn der Wert zwischen "min" und "max" liegt* • *"min", wenn der Wert kleiner "min" ist* • *"max", wenn der Wert größer "max" ist*

4.2.2 Consolen Funktionen

native print(const string[]);

druckt den angegebenen String auf die Standardausgabe

Parameter	*Erklärung*
string	*die auszugebende Zeichenfolge. Diese darf auch Escape-Sequenzen enthalten.*

	Erklärung
Rückgabewert	*OK*

native printf(const format[], {Float,Fixed,_}:...);

druckt den übergebenen Format-String auf die Standardausgabe. Die Arbeitsweise der Funktionen entspricht jener der Standard ANSI-C Implementierung.

Parameter	*Erklärung*
format[]	*die zu verwendende Format-Zeichenkette*

	Erklärung
Rückgabewert	• *Anzahl der gedruckten Zeichen* • *ERROR, falls nicht erfolgreich*

4.2.3 rapidM2M Funktionen

4.2.3.1 Arrays mit symbolischen Indizes

TrM2M_Id

Informationen zur Identifikation des Moduls/Geräts

```
// string     rapidM2M Modulidentifikation (z.B. "rapidM2M IoT-Box
            HW1.1")
// module     rapidM2M Modultyp (z.B. "IoT-Box")
// hwmajor    Hardware: Hauptversionsnummer
// hwminor    Hardware: Nebenversionsnummer
// sn         Geräte-Seriennummer (binär) im BIG Endian Format
//            Bsp.: "010146AF251CED1C" --> "01" in sn{0}, "1C" in sn{7}
// fwmajor    Firmware: Hauptversionsnummer
// fwminor    Firmware: Nebenversionsnummer
// ctx        Messstellenbezeichnung (Kontext)
//            Leerstring, wenn kein Kontext vorhanden ist

#define TrM2M_Id[  .string{50}, .module{10}, .hwmajor, .hwminor,
                   .sn{8}, .fwmajor, .fwminor, .ctx{50}             ]
```

TrM2M_GSMPos

Informationen über eine GSM-Zelle im Empfangsbereich

```
// mcc        MCC (Mobile Country Code) der GSM-Zelle
// mnc        MNC (Mobile Network Code) der GSM-Zelle
// lac        LAC (Location Area Code) der GSM-Zelle
// cellid     Cell ID der GSM-Zelle
// rssi       empfangene GSM-Feldstärke [dBm] für die GSM-Zelle
// ta         TA (Timing Advance) der GSM-Zelle (dzt. immer auf 0)

#define TrM2M_GSMPos[.mcc, .mnc, .lac, .cellid, .rssi, .ta]
```

TrM2M_GSMInfo

Informationen zum GSM-Modem, SIM-Chip sowie dem bei der letzten Verbindung verwendeten GSM-Netz

```
// cgmi       Manufacturer Identification des Modems
// cgmm       Modem Modellinformation
// cgmr       Modem Revisionsinformation
// imei       International Mobile Equipment Identity des Modems
// imsi       International Mobile Subscriber Identity des SIM-Chips, der für
//            die letzte Verbindung verwendet wurde
//            Leerstring, wenn noch keine Verbindung stattgefunden hat
// iccid      Integrated Circuit Card Identifier des SIM-Chips, der für die
//            letzte Verbindung verwendet wurde
//            Leerstring, wenn noch keine Verbindung stattgefunden hat
// mcc        MCC (Mobile Country Code) des Netzes, das für die letzte
//            Verbindung verwendet wurde
//            0, wenn noch keine Verbindung stattgefunden hat
// mnc        MNC (Mobile Network Code) des Netzes, das für die letzte
//            Verbindung verwendet wurde
//            0, wenn noch keine Verbindung stattgefunden hat
// simstate   Aktueller SIM-Status (siehe "SIM-Status" im Kapitel
//            "Konstanten" auf Seite 32)

#define TrM2M_GSMInfo[  .cgmi{20}, .cgmm{20}, .cgmr{20}, .imei{16}, .imsi{16},
                        .iccid{21}, .mcc, .mnc, .simstate                    ]
```

4.2.3.2 Konstanten

Returncodes für allgemeine Zwecke

```
OK                      = 0,
ERROR                   = -1,
ERROR_PARAM             = -2,  // Parameter error
ERROR_UNKNOWN_HDL       = -3,  // Unknown handler, handle or resource error
ERROR_ALREADY_SUBSCRIBED = -4,  // Already subscribed service or resource error
ERROR_NOT_SUBSCRIBED    = -5,  // Not subscribed service error
ERROR_FATAL             = -6,  // Fatal error
ERROR_BAD_HDL           = -7,  // Bad handle or resource error
ERROR_BAD_STATE         = -8,  // Bad state error
ERROR_PIN_KO            = -9,  // Bad PIN state error
ERROR_NO_MORE_HANDLES   = -10, /* The service subscription maximum capacity is
                                  reached */
ERROR_DONE              = -11, /* The required iterative process is now
                                  terminated */
ERROR_OVERFLOW          = -12, /* The required operation has exceeded the
                                  function capabilities */
ERROR_NOT_SUPPORTED     = -13, /* An option, required by the function, is not
                                  enabled on the CPU, the function is not
                                  supported in this configuration */
ERROR_NO_MORE_TIMERS    = -14, /* The function requires a timer subscription,
                                  but no more timer resources are available */
ERROR_NO_MORE_SEMAPHORES = -15, /* The function requires a semaphore allocation,
                                  but there are no more semaphore resources */
ERROR_SERVICE_LOCKED    = -16, /* The function was called from a low or high
                                  level interrupt handler (the function is
                                  forbidden in this case) */
ERROR_MEM               = -100,// error allocating memory
ERROR_SIM_STATE         = -101,// SIM state error
```

SIM-Status

```
//Verbindung kann per Script ausgelöst werden
RM2M_SIM_STATE_NONE       = 0,   //Initialzustand
RM2M_SIM_STATE_PRODUCTION = 1,   //Neu produziertes Gerät liegt auf Lager
RM2M_SIM_STATE_HOT        = 2,   //Gültiger Vertrag

//Auslösen der Verbindung per Script nicht möglich
RM2M_SIM_STATE_COLD       = 3,   //Vertragsende oder Fair-Use Verletzung
RM2M_SIM_STATE_DISCARDED  = 4,   //Gerät wurde außer Dienst gestellt
```

Einstellung der Signalrichtung der GPIOs

```
RM2M_GPIO_INPUT    = 0,          // Eingang
RM2M_GPIO_OUTPUT   = 1,          // Ausgang
```

Singnalpegel der GPIOs

```
RM2M_GPIO_LOW      = 0,          // Signalpegel "low"
RM2M_GPIO_HIGH     = 1,          // Signalpegel "high"
```

Konfiguration von Clock-Polarität und Clock-Phase des SPI-Interfaces

Konfigurationsflags für die Funktion rM2M_SpiInit()

```
RM2M_SPI_CLKPOL    = 0b00000001, // Clock-Polarität: Idle "high"
RM2M_SPI_CLKPHA    = 0b00000010, /* Clock-Phase: Daten werden bei der ersten
                                    Flanke übernommen. */
```

Konfiguration des UART-Interfaces

Konfigurationsflags für die Funktion rM2M_UartInit()

```
RM2M_UART_1_STOPBIT   = 0b00000001, // 1 Stopbit
RM2M_UART_2_STOPBIT   = 0b00000010, // 2 Stopbit
RM2M_UART_PARITY_NONE = 0b00000000, // keine Parität
RM2M_UART_PARITY_ODD  = 0b00000100, // ungerade Parität
RM2M_UART_PARITY_EVEN = 0b00001000, // gerade Parität
RM2M_UART_7_DATABIT   = 0b00000000, // 7 Datenbits
RM2M_UART_8_DATABIT   = 0b00010000, // 8 Datenbits
RM2M_UART_FLOW_NONE   = 0b00000000, // keine Flusskontrolle
RM2M_UART_FLOW_RTSCTS = 0b01000000, // RTS/CTS Handshake
```

Verbindungsflags

Steuerflags für die Funktion rM2M_TxStart()

```
RM2M_TX_POSUPDATE  = 0b00000001, /* Update der GSM-Positionsdaten beim
                                    Verbindungsaufbau */
```

Kommunikationsmodi

Kommunikationsmodi für die Funktion rM2M_TxSetMode()

```
RM2M_TXMODE_TRIG   = 0,          // Intervall
RM2M_TXMODE_WAKEUP = 1,          // Intervall & Wakeup
RM2M_TXMODE_ONLINE = 2,          // Online
```

Verbindungsstatus

Rückgabewerte der Funktion rM2M_TxGetStatus()

```
RM2M_TX_FAILED     = 0b00000001, // Verbindungsaufbau fehlgeschlagen
RM2M_TX_ACTIVE     = 0b00000010, // GPRS-Verbindung besteht
RM2M_TX_STARTED    = 0b00000100, // Verbindungsaufbau gestartet
RM2M_TX_RETRY      = 0b00001000, // Wartezeit bis zum Retry
RM2M_TX_WAKEUPABLE = 0b00010000, // Modem ins GSM-Netz eingebucht
```

Konfigurierungsflags für die Funktion rM2M_Pack()

```
RM2M_PACK_GET      = 0b00000001, // Wert soll gelesen werden (Get Packed)
RM2M_PACK_BE       = 0b00000010, // "Big Endian"-Format verwenden
RM2M_PACK_U8       = 0b00010000, // 8-Bit Unsigned
RM2M_PACK_S8       = 0b10010000, // 8-Bit Signed
RM2M_PACK_U16      = 0b00100000, // 16-Bit Unsigned
RM2M_PACK_S16      = 0b10100000, // 16-Bit Signed
RM2M_PACK_U32      = 0b01000000, // 32-Bit Unsigned
RM2M_PACK_S32      = 0b11000000, // 32-Bit Signed
RM2M_PACK_F32      = 0b01000000, // 32-Bit Float
```

Konfigurierungsflags für die Funktion rM2M_CfgInit()

```
RM2M_CFG_VOLATILE  = 0b00000001, // flüchtige Speicherung (RAM)
```

Indizes der Registrierungsspeicherblöcke

auf die mittels der Funktionen "rM2M_RegGetString()", "rM2M_RegGetValue()", "rM2M_RegSetString()" und "rM2M_RegSetValue()" zugegriffen werden kann.

```
//Systemspezifische Daten
RM2M_REG_SYS_OTP   = 0, // einmalig im Zuge der Produktion (Empfehlung)
RM2M_REG_SYS_FLASH = 1, // im Betrieb veränderbar (nicht durch Script)

//Applikationsspezifische Daten
RM2M_REG_APP_OTP   = 2, // einmalig im Zuge der Produktion (Empfehlung)
RM2M_REG_APP_FLASH = 3, // im Betrieb veränderbar (auch durch Script)

//Anzahl der Registrierungsspeicherblöcke
RM2M_REG_NUM_REGS  = 4,
```

NMEA Fehlercodes

Fehlercodes der Funktion rM2M_SetPosNMEA()

```
RM2M_NMEA_ERR_DATATYPE  = -2, // Datentyp (z.B. $GGSA) wird nicht unterstützt.
RM2M_NMEA_ERR_SENTENCE  = -3, // Sentence ungültig (z.B. Checksum Fehler)
RM2M_NMEA_ERR_LATITUDE  = -4, // geographische Breite ungültig
RM2M_NMEA_ERR_LONGITUDE = -5, // geographische Länge ungültig
RM2M_NMEA_ERR_ALTITUDE  = -6, // Höhe über dem Meeresspiegel ungültig
RM2M_NMEA_ERR_SAT_USED  = -7, // Anzahl der verwendeten Satelliten ist ungültig.
RM2M_NMEA_ERR_QUAL      = -8, // GPS-Qualitätsangabe wird nicht unterstützt.
```

4.2.3.3 Callback Funktionen

public func(const data{}, len);

vom Script-Entwickler bereitzustellende Funktion, die beim Empfang von Zeichen über die UART-Schnittstelle aufgerufen wird.

Parameter	*Erklärung*
data	*Array, das die empfangenen Daten enthält*
len	*Anzahl der empfangenen Bytes*

public func(cfg);

vom Script-Entwickler bereitzustellende Funktion, die aufgerufen wird, wenn sich einer der Konfigurationsspeicherblöcke geändert hat

Parameter	*Erklärung*
cfg	*Nummer des Konfigurationsspeicherblocks, beginnend mit 0 für den ersten Speicherblock, der geändert wurde*

public func(reg);

vom Script-Entwickler bereitzustellende Funktion, die aufgerufen wird, wenn sich die Registrierung geändert hat

Parameter	*Erklärung*
reg	*Index des Registrierungsspeicherblocks (siehe "Indizes der Registrierungsspeicherblöcke" im Kapitel "Konstanten" auf Seite 32), der geändert wurde*

public func(const SmsTel[], const SmsText[]);

vom Script-Entwickler bereitzustellende Funktion, die aufgerufen wird, wenn eine SMS empfangen wurde

Parameter	*Erklärung*
SmsTel	*String, der die Telefonnummer des Absenders der SMS enthält*
SmsText	*String, der den Inhalt der SMS enthält*

4.2.3.4 Funktionen

native rM2M_GetId(id[TrM2M_Id], len=sizeof id);

liefert die Informationen zur Identifikation des Moduls/Geräts

Parameter	*Erklärung*
id	*Struktur zur Aufnahme der Informationen zur Identifikation des Moduls/Geräts (siehe "TrM2M_Id" im Kapitel "Arrays mit symbolischen Indizes" auf Seite 30)*
len	*Größe (in Cells) der Struktur zur Aufnahme der Informationen - OPTIONAL*

	Erklärung
Rückgabewert	• *Verwendete Größe (in Cells) der Struktur zur Aufnahme der Informationen* • *ERROR, wenn Adresse und/oder Länge der id-Struktur ungültig sind (außerhalb des Skript-Datenspeichers)* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)* ***Hinweis:*** *Die Firmware des Moduls/Geräts erkennt, wenn ein Script verwendet wird, bei dem die Funktion nur einen Übergabeparameter (Verwendung eines älteren Include-Files) besitzt und liefert aus Kompatibilitätsgründen "OK" anstelle der Göße der Struktur zur Aufnahme der Informationen zurück.*

native rM2M_GetTime(&hour=0, &minute=0, &second=0, timestamp=0);

Wurde kein Timestamp übergeben (timestamp=0), wird die aktuelle Systemzeit (in UTC) in Stunden / Minuten / Sekunden konvertiert. Andernfalls wird der übergebene Timestamp in Stunden / Minuten / Sekunden konvertiert.

Parameter	*Erklärung*
hour	*Variable zur Aufnahme der Stunden - OPTIONAL*
minute	*Variable zur Aufnahme der Minuten - OPTIONAL*
second	*Variable zur Aufnahme der Sekunden - OPTIONAL*
timestamp	*Zeitstempel, der konvertiert werden soll* *= 0: Es wird die aktuelle Systemzeit (in UTC) konvertiert.* *› 0: Es wird der übergebene Zeitstempel konvertiert.* *(Der Zeitstempel muss in Sekunden seit 31.12.1999 angegeben werden.)*

	Erklärung
Rückgabewert	• *timestamp = 0: Sekunden seit 31.12.1999 (aktuelle Systemzeit in UTC)* • *timestamp › 0: Der übergebene Zeitstempel wird zurückgegeben.*

native rM2M_GetDate(&year=0, &month=0, &day=0, timestamp=0);

Wurde kein Timestamp übergeben (timestamp=0), wird für die aktuelle Systemzeit (in UTC) das Datum (Jahr, Monat, Tag) ermittelt. Andernfalls wird für den übergebenen Timestamp das Datum (Jahr, Monat, Tag) ermittelt.

Parameter	*Erklärung*
year	*Variable zur Aufnahme des Jahres - OPTIONAL* ***Hinweis:*** *Die Angabe des Jahres erfolgt relativ zum Jahr 2000, d.h. für das Jahr 2014 wird der Wert 14 retourniert.*
month	*Variable zur Aufnahme des Monats - OPTIONAL*
day	*Variable zur Aufnahme des Tags - OPTIONAL*
timestamp	*Zeitstempel für den das Datum ermittelt werden soll* *= 0: Es wird das Datum für die aktuelle Systemzeit (in UTC) ermittelt.* *› 0: Es wird das Datum für den übergebene Zeitstempel ermittelt.* *(Der Zeitstempel muss in Sekunden seit 31.12.1999 angegeben werden.)*

	Erklärung
Rückgabewert	• *timestamp = 0: Sekunden seit 31.12.1999 (aktuelle Systemzeit in UTC)* • *timestamp › 0: Der übergebene Zeitstempel wird zurückgegeben.*

native rM2M_GetTimezoneOffset();

liefert die Differenz (in Sekunden) zwischen Systemzeit (UTC) und der für die Messstelle am C-Control-Server konfigurierten lokalen Zeit. Dadurch kann im Skript die lokale Zeit bestimmt werden, indem diese Differenz zur Systemzeit (UTC) addiert wird. Der Offsetwert wird vom C-Control-Server entsprechend der eingestellten Zeitzone (inkl. Sommer-/Winterzeit) gebildet und bei jeder Verbindung mit dem Gerät synchronisiert.

Bsp.: Für die Messstelle wird die mitteleuropäische Zeit (MEZ = UTC+1) verwendet -› Offset = 3600sec.

	Erklärung
Rückgabewert	*Offsetwert in Sekunden*

native rM2M_DoW(timestamp);

berechnet den Wochentag aus einem gegebenen Timestamp

Parameter	*Erklärung*
timestamp	*Zeitstempel des zu berechnenden Tages*

	Erklärung
Rückgabewert	*Wochentag, 0=Montag ... 6=Sonntag*

native rM2M_TimerAdd(funcidx);

erzeugt einen neuen 1s Timer

Parameter	*Erklärung*
funcidx	*Index der öffentlichen Funktion, die nach Ablauf des Timers aufgerufen werden soll* *Typ der Funktion: public func();*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR, wenn kein gültiger Index übergeben wurde, bei einem internen Fehler oder wenn keine weiteren Timer mehr angelegt werden können (maximale Anzahl erreicht)* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_TimerRemove(funcidx);

entfernt einen 1s Timer

Parameter	*Erklärung*
funcidx	*Index der öffentlichen Funktion des zu entfernenden Timers* *Typ der Funktion: public func();*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR, wenn kein gültiger Index übergeben wurde oder bei einem internen Fehler* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_TimerAddExt(funcidx, bool:cyclic, time);

erzeugt einen neuen ms Timer

Parameter	*Erklärung*
funcidx	*Index der öffentlichen Funktion, die nach Ablauf des Timers aufgerufen werden soll* *Typ der Funktion: public func();*
cyclic	*Einstellung für das Verhalten nach Ablauf des Timerintervalls:* *true: Der Timer soll nach dem Ablauf des Intervalls neu gestartet werden.* *false: Der Timer wird nach Ablauf des Intervalls gestoppt.*
time	*Timerintervall in Millisekunden*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR, wenn kein gültiger Index übergeben wurde, bei einem internen Fehler oder wenn keine weiteren Timer mehr angelegt werden können (maximale Anzahl erreicht)* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_TimerRemoveExt(funcidx);

entfernt einen ms Timer

Parameter	*Erklärung*
funcidx	*Index der öffentlichen Funktion des zu entfernenden Timers* *Typ der Funktion: public func();*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR, wenn kein gültiger Index übergeben wurde oder bei einem internen Fehler* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_GpioDir(gpio, dir);

setzt die Signalrichtung für einen GPIO

Parameter	*Erklärung*
gpio	*Nummer des GPIOs, beginnend mit 0 für GPIO1*
dir	*Einstellung der Signalrichtung:* *RM2M_GPIO_INPUT : Eingang* *RM2M_GPIO_OUTPUT : Ausgang*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_GpioSet(gpio, level);

setzt das Ausgangslevel eines GPIOs, der als Ausgang konfiguriert wurde

Parameter	*Erklärung*
gpio	*Nummer des GPIOs, beginnend mit 0 für GPIO1*
dir	*Angabe des auszugebenden Levels:* *RM2M_GPIO_LOW : Ausgangslevel auf "low"* *RM2M_GPIO_HIGH : Ausgangslevel auf "high"*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR_BAD_STATE, wenn der betreffende GPIO nicht als Ausgang konfiguriert wurde* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_GpioGet(gpio);

liest das Signallevel eines GPIOs, der als Eingang konfiguriert wurde

Parameter	*Erklärung*
gpio	*Nummer des GPIOs, beginnend mit 0 für GPIO1*

	Erklärung
Rückgabewert	• *RM2M_GPIO_LOW für "low" am Eingang* • *RM2M_GPIO_HIGH für "high" am Eingang* • *ERROR_BAD_STATE, wenn der betreffende GPIO nicht als Eingang konfiguriert wurde* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

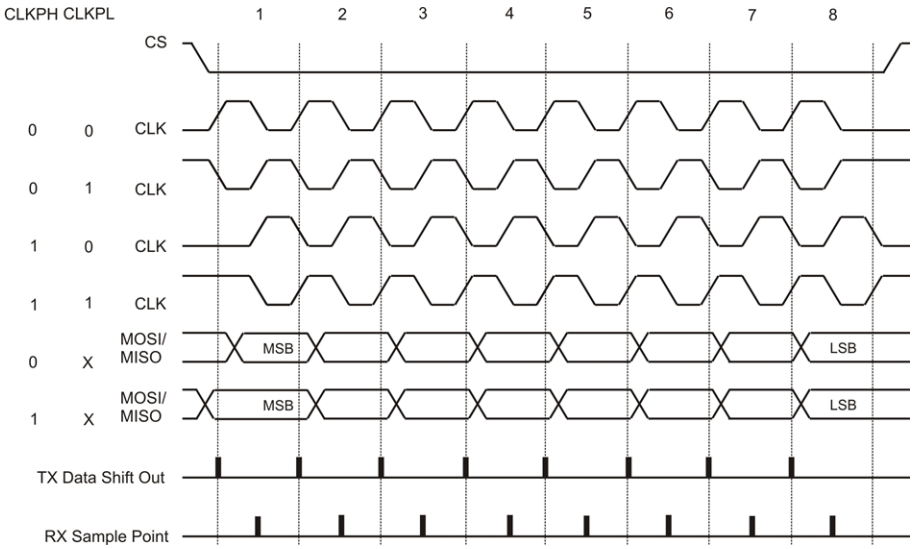
native rM2M_SpiInit(spi, clock, config);

initialisiert das SPI-Interface

Parameter	*Erklärung*
spi	*Nummer des SPI-Interfaces, beginnend mit 0 für SPI1*
clock	*zu verwendende Clockfrequenz in Hz. Bitte beachten Sie die für das verwendete Modul gültigen Grenzwerte (siehe "Technische Daten" auf Seite 125).*
config	*Konfiguration von Clock-Polarität und Clock-Phase* *Bit0: Clock-Polarität* *0 = Idle "low"* *1 = Idle "high"* *Bit1: Clock-Phase* *0 = Daten werden bei der ersten Flanke ausgegeben.* *1 = Daten werden bei der ersten Flanke übernommen.* ***Hinweis:** Sie können für diesen Parameter auch die vordefinierten Konstanten verwenden (siehe "Konfiguration von Clock-Polarität und Clock-Phase des SPI-Interface" im Kapitel "Konstanten" auf Seite 32). Die Konstanten lassen sich auch durch "oder"-Verknüpfung kombinieren.*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR, wenn die Nummer des SPI-Interfaces ungültig ist* • *ERROR_ALREADY_SUBSCRIBED, wenn das SPI-Interface bereits initialisiert wurde* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

***Hinweis:** Ergänzende Erklärung zur Konfiguration von Clock-Polarität und Clock-Phase*



native rM2M_SpiClose(spi);

schließt das SPI-Interface

Parameter	*Erklärung*
spi	*Nummer des SPI-Interfaces, beginnend mit 0 für SPI1*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR, wenn die Nummer des SPI-Interfaces ungültig ist* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_SpiCom(spi, data{}, txlen, rxlen);

führt eine asynchrone SPI Kommunikation aus. Zuerst werden Daten versendet, danach Daten empfangen.

Parameter	*Erklärung*
spi	*Nummer des SPI-Interfaces, beginnend mit 0 für SPI1*
data	*Array, in dem zunächst die zu versendenden Daten gespeichert sein müssen. Wurden die Daten versendet, wird das Array als Speicher für die zu empfangenden Daten verwendet.*
txlen	*Anzahl der zu sendenden Bytes*
rxlen	*Anzahl der zu empfangenden Bytes*

	Erklärung

Rückgabewert	• *OK, wenn erfolgreich*
	• *ERROR, wenn einer der folgenden Fehler auftritt*
	  ◦ *Anzahl der zu sendenden Bytes › 255*
	  ◦ *Anzahl der zu empfangenden Bytes › 255*
	• *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_I2cInit(i2c, clock, config);

initialisiert das I²C-Interface

Parameter	*Erklärung*
i2c	*Nummer des I²C-Interfaces, beginnend mit 0 für I2C1*
clock	*zu verwendende Clockfrequenz in Hz. Bitte beachten Sie die für das verwendete Modul gültigen Grenzwerte (siehe "Technische Daten" auf Seite 125).*
config	*reserviert für Erweiterungen*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR, wenn die Nummer des I²C-Interfaces ungültig ist* • *ERROR_ALREADY_SUBSCRIBED, wenn das I²C-Interface bereits initialisiert wurde* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_I2cClose(i2c);

schließt das I²C-Interface

Parameter	*Erklärung*
i2c	*Nummer des I²C-Interfaces, beginnend mit 0 für I2C1*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR, wenn die Nummer des I²C-Interfaces ungültig ist* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_I2cCom(i2c, adr, data{}, txlen, rxlen);

führt eine I²C Kommunikation aus. Zuerst werden Daten versendet, danach Daten empfangen.

Parameter	*Erklärung*
i2c	*Nummer des I²C-Interfaces, beginnend mit 0 für I2C1*
adr	*Adresse des I²C Slaves (Bit7-Bit1, Bit0 unused)*
data	*Array, in dem zunächst die zu versendenden Daten gespeichert sein müssen. Wurden die Daten versendet, wird das Array als Speicher für die zu empfangenden Daten verwendet.*
txlen	*Anzahl der zu sendenden Bytes*
rxlen	*Anzahl der zu empfangenden Bytes*

	Erklärung

Rückgabewert	• *OK, wenn erfolgreich*
	• *ERROR, wenn einer der folgenden Fehler auftritt*
	  ◦ *Anzahl der zu sendenden Bytes › 255*
	  ◦ *Anzahl der zu empfangenden Bytes › 255*
	• *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_UartInit(uart, baudrate, mode, funcidx);

initialisiert das UART-Interface

Parameter	*Erklärung*
uart	*Nummer des UART-Interfaces, beginnend mit 0 für UART1*
baudrate	*zu verwendende Baudrate. Bitte beachten Sie die für das verwendete Modul gültigen Grenzwerte (siehe "Technische Daten" auf Seite 125).*
mode	*Bit 0...1* *1 = 1 Stopbit* *2 = 2 Stopbit* *Bit 2...3* *0 = keine Parität* *1 = ungerade Parität* *2 = gerade Parität* *Bit 4...5* *0 = 7 Datenbits* *1 = 8 Datenbits* *Bit 6...7* *0 = keine Flusskontrolle* *1 = RTS/CTS Handshake* ***Hinweis:** Sie können für diesen Parameter auch die vordefinierten Konstanten verwenden (siehe "Konfiguration des UART-Interface" im Kapitel "Konstanten" auf Seite 32). Die Konstanten lassen sich auch durch "oder"-Verknüpfung kombinieren.*
funcidx	*Index der öffentlichen Funktion für den UART-Zeichenempfang* *Typ der Funktion: public func(const data{}, len);*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_UartClose(uart);

schließt das UART-Interface

Parameter	*Erklärung*
uart	*Nummer des UART-Interfaces, beginnend mit 0 für UART1*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_UartWrite(uart, const data{}, len);

versendet Daten über das angegebene UART-Interface

Parameter	*Erklärung*
uart	*Nummer des UART-Interfaces, beginnend mit 0 für UART1*
data	*Array, das die zu sendenden Daten enthält*
len	*Anzahl der zu sendenden Bytes*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR, wenn die Anzahl der zu sendenden Bytes › 256* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_TxStart(flags=0);

löst eine Verbindung zum Server aus

Parameter	*Erklärung*
flags	*Wenn Bito (RM2M_TX_POSUPDATE) gesetzt ist, erfolgt auch ein Update der GSM-Positionsdaten .*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR, wenn der Verbindungsaufbau aufgrund zu niedriger Versorgungsspannung nicht möglich ist* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_TxSetMode(mode);

setzt die zu verwendende Verbindungsart

Parameter	*Erklärung*
mode	*zu verwendende Verbindungsart:* *RM2M_TXMODE_TRIG: Die Verbindung erfolgt beim Aufruf der Funktion "rM2M_TxStart()"* *RM2M_TXMODE_WAKEUP: Die Verbindung erfolgt wie im Modus "intervall" beim Aufruf der Funktion "rM2M_TxStart()". Zusätzlich kann das Gerät über den Server dazu veranlasst werden, sofort eine Verbindung aufzubauen (siehe "Benutzerhandbuch für C-Control -Server " 206.886). Dazu bucht sich das Gerät unverzüglich ins GSM-Netz ein sobald dieser Modus gesetzt wurde .*  *RM2M_TXMODE_ONLINE: Das Gerät trennt die Verbindung nicht und übermittelt kontinuierlich die Messdaten.Der Verbindungsaufbau erfolgt unverzüglich sobald dieser Modus gesetzt wurde. Ein Aufruf der Funktion "rM2M_TxStart()" ist nicht erforderlich.*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR, wenn der Verbindungsaufbau aufgrund zu niedriger Versorgungsspannung nicht möglich ist* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

Hinweis: *Ergänzende Erklärung zu den Verbindungsarten*

Verbindungsart	*Energieverbrauch*	*Datenvolumen*	*Reaktionszeit*
online			
Intervall & Wakeup			
Intervall			

native rM2M_TxGetStatus();

liefert den aktuellen Verbindungsstatus

	Erklärung	
Rückgabewert	*Bito (RM2M_TX_FAILED):*	*gesetzt, wenn der letzte GPRS-Verbindungsaufbau fehlgeschlagen ist*
	Bit1 (RM2M_TX_ACTIVE):	*gesetzt, wenn eine GPRS-Verbindung besteht*
	Bit2 (RM2M_TX_STARTED):	*gesetzt, wenn der Verbindungsaufbau gestartet wurde*
	Bit3 (RM2M_TX_RETRY):	*gesetzt, während der Wartezeit bis zum erneuten automatischen Retry bei Verbindungsproblemen*
	Bit4 (RM2M_TX_WAKEUPABLE):	*gesetzt, wenn das Modem ins GSM-Netz eingebucht ist (Wakeup ist möglich)*

native rM2M_GSMGetRSSI();

liefert die GSM-Signalstärke

	Erklärung
Rückgabewert	*Signalstärke in [dBm] (maximaler Wertebereich: -128...127)* *GSM-Werte sind im Bereich -113 .. -51 dBm.*

native rM2M_RecData(timestamp, const data{}, len);

speichert einen Datensatz im internen Flash-Speicher. Benutzen Sie die Funktionen "rM2M_Pack", "rM2M_SetPacked" oder "rM2M_SetPackedB", um den Datenbereich zu erzeugen.

Parameter	*Erklärung*
timestamp	*Zeitstempel, der für die Aufzeichnung verwendet werden soll* *= 0: Die aktuelle Systemzeit wird als Zeitstempel verwendet.* *› 0: Der übergebene Zeitstempel wird verwendet.* *(Der Zeitstempel muss in Sekunden seit 31.12.1999 angegeben werden)*
data	*Array, das die zu speichernden Daten enthält*
len	*Anzahl der zu speichernden Bytes (max. 1024 Byte)*

	Erklärung

Rückgabewert	• *OK, wenn erfolgreich*
	• *-2, wenn die Datenspeicherung aktuell nicht möglich ist, weil der interne Speicher reorganisiert wird. Die Daten müssen im Skript zwischengespeichert und zu einem späteren Zeitpunkt erneut gesichert werden.*
	• *ERROR, wenn einer der folgenden Fehler auftritt*
	  • *Speicherbereich (data{}, len) ist ungültig.*
	  • *Mehr als 10 Aufrufe in einem Skriptdurchlauf*
	  • *Anzahl der zu speichernden Bytes › 1024 Byte*
	  • *FLASH-Schreibvorgang nicht erfolgreich*
	  • *Übergabeparameter timestamp liegt mehr als 5 Minuten in der Zukunft*
	• *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_SetPacked(data{}, pos, &{Float,Fixed,_}:value, size=4, bool:bigendian=false);

schreibt den übergebenen Wert an die angegebene Position in ein Array

Parameter	*Erklärung*
data	*Array, das als Datenbereich für einen Datensatz oder eine Konfiguration verwendet werden soll*
pos	*Byteoffset innerhalb des Arrays zur Bestimmung der Position, an die der Wert geschrieben werden soll*
value	*Wert, der in das Array geschrieben werden soll*
size	*Anzahl der Bytes, die für den zu schreibenden Wert verwendet werden sollen*
bigendian	*Einstellung, für die zu verwendende Byte-Reihenfolge beim Schreiben des Werts:* *true: "Big Endian" wird verwendet* *false: "Little Endian" wird verwendet*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

Hinweis: *Ergänzende Erklärung zur Byte-Reihenfolge:*

Im folgenden Beispiel wird die Ganzzahl 439.041.101 als 32-Bit-Integer-Wert ab Speicheradresse 10000 gespeichert.

	Big Endian			*Little Endan*		
Adressen	*Hex*	*Dez*	*Binär*	*Hex*	*Dez*	*Binär*
10000	*1A*	*26*	*00011010*	*4D*	*77*	*01001101*
10001	*2B*	*43*	*00101011*	*3C*	*60*	*00111100*
10002	*3C*	*60*	*00111100*	*2B*	*43*	*00101011*
10003	*4D*	*77*	*01001101*	*1A*	*26*	*00011010*

native rM2M_SetPackedB(data{}, pos, const block[], size);

schreibt den übergebenen Datenblock an die angegebene Position in ein Array

Parameter	*Erklärung*
data	*Array, das als Datenbereich für einen Datensatz oder eine Konfiguration verwendet werden soll*
pos	*Byteoffset innerhalb des Arrays zur Bestimmung der Position, an die der Datenblock geschrieben werden soll*
block	*Datenblock, der in das Array geschrieben werden soll*
size	*Anzahl der Bytes, die vom Datenblock in das Array geschrieben werden sollen*

native rM2M_GetPacked(const data{}, pos, &{Float,Fixed,_}:value, size=4, bool:bigendian=false);

liefert den Wert, der sich an der angegebenen Position im einem Array befindet

Parameter	*Erklärung*
data	*Array, das als Datenbereich für einen Datensatz oder eine Konfiguration verwendet werden soll*
pos	*Byteoffset innerhalb des Arrays zur Bestimmung der Position, von der die Daten gelesen werden sollen*
value	*Variable zur Aufnahme der zu lesenden Daten*
size	*Anzahl der Bytes, die zu lesen sind*
bigendian	*Gibt an, wie die gepackten Daten zu interpretieren sind:* *true: Die Daten sind im "Big Endian"-Format im Array gespeichert.* *false: Die Daten sind im "Little Endian"-Format im Array gespeichert.*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

Hinweis: *Ergänzende Erklärung zur Byte-Reihenfolge:*

Im folgenden Beispiel wird die Ganzzahl 439.041.101 als 32-Bit-Integer-Wert ab Speicheradresse 10000 gespeichert.

	Big Endian			*Little Endan*		
Adressen	*Hex*	*Dez*	*Binär*	*Hex*	*Dez*	*Binär*
10000	*1A*	*26*	*00011010*	*4D*	*77*	*01001101*
10001	*2B*	*43*	*00101011*	*3C*	*60*	*00111100*
10002	*3C*	*60*	*00111100*	*2B*	*43*	*00101011*
10003	*4D*	*77*	*01001101*	*1A*	*26*	*00011010*

native rM2M_GetPackedB(const data{}, pos, block[], size);

liest einen Datenblock, der sich an der angegebenen Position in einem Array befindet

Parameter	*Erklärung*
data	*Array, das als Datenbereich für einen Datensatz oder eine Konfiguration verwendet werden soll*
pos	*Byteoffset innerhalb des Arrays zur Bestimmung der Position, von der die Daten gelesen werden sollen*
block	*Array zur Aufnahme der zu lesenden Daten*
size	*Anzahl der Bytes, die zu lesen sind*

native rM2M_Pack(const data{}, pos, &{Float,Fixed,_}:value, type);

Funktion für den Zugriff auf gepackte Daten. Wurde das Bito (RM2M_PACK_GET) des Parameters "type" gesetzt, liefert die Funktion den Wert, der sich an der angegebenen Position im Array befindet. Andernfalls schreibt die Funktion den übergebenen Wert an die angegebene Position ins Array.

Parameter	*Erklärung*
data	*Array mit den gepackten Inhalten* *Set Packed: Array, in das der Wert geschrieben werden soll* *Get Packed: Array, aus dem der Wert gelesen werden soll*
pos	*Byteoffset innerhalb des Arrays* *Set Packed: Position, an die der Wert geschrieben werden soll* *Get Packed: Position, von der der Wert gelesen werden soll*
value	*Set Packed: Wert, der in das Array geschrieben werden soll* *Get Packed: Wert, der aus dem Array gelesen werden soll*
type	*Konfigurierungsflags für die Funktion* *Bito: Auswahl Set Packed / Get Packed* *0 = Wert soll geschrieben werden* *1 = Wert soll gelesen werden* *Bit1: Byte-Reihenfolge* *0 = "Little Endian"-Format* *1 = "Big Endian"-Format* *Bit2...3* *reserviert für Erweiterungen* *Bit4...7: Datentyp* *1 = 8-Bit Unsigned* *2 = 16-Bit Unsigned* *4 = 32-Bit Unsigned / 32-Bit Float* *9 = 8-Bit Signed* *10 = 16-Bit Signed* *12 = 32-Bit Signed* ***Hinweis:** Sie können für diesen Parameter auch die vordefinierten Konstanten verwenden (siehe "Konfigurierungsflags für die Funktion rM2M_Pack()" im Kapitel "Konstanten" auf Seite 32). Die Konstanten lassen sich auch durch "oder"-Verknüpfung kombinieren.*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_CfgInit(cfg, flags);

legt die Konfiguration für einen Konfigurationsspeicherblock fest. Ein Aufruf der Funktion ist nur notwendig, wenn eines der Konfigurationsflags gesetzt werden soll.

Parameter	*Erklärung*
cfg	*Nummer des Konfigurationsspeicherblocks, beginnend mit 0 für den ersten Speicherblock. Das Gerät verfügt über 10 voneinander unabhängige Speicherblöcke.*
flags	*Zu setzende/löschende Konfigurationsflags* *Bito: Art der Speicherung* *0 (default) = nichtflüchtig im FLASH gespeichert* *RM2M_CFG_VOLATILE = flüchtig im RAM gespeichert*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *< OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

Hinweis: *Ergänzende Erklärung zur Art der Speicherung:*

Wenn Bit 0 nicht gesetzt wurde (default), wird beim Aufruf der Funktion "rM2M_CfgWrite" der Konfigurationsspeicherblock nicht flüchtig im FLASH gespeichert.

Wenn Bito gesetzt wurde (Bito = RM2M_CFG_VOLATILE), wird beim Aufruf der Funktion "rM2M_CfgWrite" der Konfigurationsspeicherblock flüchtig im RAM gespeichert. Diese Option ist zu empfehlen, wenn sich die Daten im Konfigurationsspeicherblock häufig ändern, da dadurch die Anzahl der Flash-Schreibzyklen reduziert wird. Um den Konfigurationsspeicherblock nicht flüchtig im FLASH zu speichern, muss die Funktion "rM2M_CfgFlush" aufgerufen werden.

native rM2M_CfgWrite(cfg, pos, const data{}, size);

speichert den übergebenen Datenblock an der angegeben Position in einem Konfigurationsspeicherblock. Bitte beachten Sie, dass der Konfigurationspeicherblock abhängig von der mit Hilfe der Funktion "rM2M_CfgInit" ausgewählten Art der Speicherung gespeichert wird, entweder flüchtig im RAM (Bito = RM2M_CFG_VOLATILE) oder nichtflüchtig im FLASH (Bito = 0, default). Der Funktion wird auch übergeben, welcher der 10 verfügbaren Speicherblocks im internen Flash-Speicher verwendet werden soll. Benutzen Sie die Funktionen "rM2M_Pack", "rM2M_SetPacked" oder "rM2M_SetPackedB", um den zu speichernden Datenblock zu erzeugen. Der Zeitstempel wird aktualisiert, wodurch der Konfigurationsspeicherblock bei der nächsten Verbindung automatisch mit dem C-Control-Server synchronisiert wird.

Parameter	*Erklärung*
cfg	*Nummer des Konfigurationsspeicherblocks, beginnend mit 0 für den ersten Speicherblock. Das Gerät verfügt über 10 voneinander unabhängige Speicherblöcke.*
pos	*Byteoffset innerhalb des Konfigurationsspeicherblocks zur Bestimmung der Position, an die die Daten geschrieben werden sollen*
data	*Array, das die Daten, die in den Konfigurationsspeicherblock geschrieben werden sollen, enthält*
size	*Anzahl der Bytes, die in den Konfigurationsspeicherblock geschrieben werden sollen*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR_MEM, wenn momentan nicht genügend temporärer Speicher (RAM) verfügbar ist. (Kann auftreten, wenn für mehrere Konfigurationsspeicherblöcke als Art der Speicherung "flüchtig im RAM" gewählt wurde.)* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_CfgFlush(cfg);

speichert den Konfigurationsspeicherblock dessen Nummer übergeben wurde nicht flüchtig im FLASH. Ein Aufruf der Funktion ist nur dann notwendig, wenn mittels der Funktion "rM2M_CfgInit" als Art der Speicherung für den betreffenden Konfigurationsspeicherblock "flüchtig im RAM (Bito = RM2M_CFG_VOLATILE)" ausgewählt wurde.

Parameter	*Erklärung*
cfg	*Nummer des Konfigurationsspeicherblocks, beginnend mit 0 für den ersten Speicherblock. Das Gerät verfügt über 10 voneinander unabhängige Speicherblöcke.*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_CfgRead(cfg, pos, data{}, size);

liest einen Datenblock von der angegeben Position aus einem Konfigurationsspeicherblock. Der Funktion wird auch übergeben, von welchem der 10 verfügbaren Speicherblocks im internen Flash-Speicher gelesen werden soll. Benutzen Sie die Funktionen "rM2M_Pack", "rM2M_GetPacked" oder "rM2M_GetPackedB", um gelesene Daten zu entpacken.

Parameter	*Erklärung*
cfg	*Nummer des Konfigurationsspeicherblocks, beginnend mit 0 für den ersten Speicherblock. Das Gerät verfügt über 10 voneinander unabhängige Speicherblöcke.*
pos	*Byteoffset innerhalb des Konfigurationsspeicherblocks zur Bestimmung der Position, von der die Daten gelesen werden soll*
data	*Array zur Aufnahme der zu lesenden Daten*
size	*Anzahl der Bytes, die aus dem Konfigurationsspeicherblock zu lesen sind*

	Erklärung
Rückgabewert	• *Größe des verwendeten Speichers im Konfigurationsspeicherblock* • *ERROR_MEM, wenn momentan nicht genügend temporärer Speicher (RAM) verfügbar ist. (Kann auftreten, wenn für mehrere Konfigurationsspeicherblöcke als Art der Speicherung "flüchtig im RAM" gewählt wurde.)* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_CfgDelete(cfg);

löscht alle Daten des übergebenen Konfigurationsspeicherblocks

Parameter	*Erklärung*
cfg	*Nummer des Konfigurationsspeicherblocks, beginnend mit 0 für den ersten Speicherblock. Das Gerät verfügt über 10 voneinander unabhängige Speicherblöcke.*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR_MEM, wenn momentan nicht genügend temporärer Speicher (RAM) verfügbar ist. (Kann auftreten, wenn für mehrere Konfigurationsspeicherblöcke als Art der Speicherung "flüchtig im RAM" gewählt wurde.)* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_CfgOnChg(funcidx);

legt die Funktion fest, die aufgerufen werden soll, wenn sich einer der Konfigurationsspeicherblöcke geändert hat

Parameter	*Erklärung*
funcidx	*Index der öffentlichen Funktion, die aufgerufen werden soll, wenn sich die Konfiguration geändert hat* *Typ der Funktion: public func(cfg);*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR, wenn kein gültiger Index einer öffentlichen Funktion übergeben wurde* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_RegGetString(reg, const name[], string[], len=sizeof string);

liest eine Zeichenkette aus einem Registrierungsspeicherblock.

Parameter	*Erklärung*
reg	*Index des Registrierungsspeicherblocks (siehe "Indizes der Registrierungsspeicherblöcke" im Kapitel "Konstanten" auf Seite 32)*
name	*Name des Eintrags*
string	*Array zur Aufnahme des zu lesenden Strings*
len	*Größe (in Cells) des übergebenen Arrays zur Aufnahme des zu lesenden Strings - OPTIONAL*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_RegGetValue(reg, const name[], &{Float,Fixed,_}:value, tag=tagof value);

liest einen Wert aus einem Registrierungsspeicherblock.

Parameter	*Erklärung*
reg	*Index des Registrierungsspeicherblocks (siehe "Indizes der Registrierungsspeicherblöcke" im Kapitel "Konstanten" auf Seite 32)*
name	*Name des Eintrags*
value	*Variable zur Aufnahme des zu lesenden Werts*
tag	*Anhand des "tag" der Variablen wird zwischen Integer oder Gleitkomma Konvertierung differenziert. - OPTIONAL*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_RegSetString(reg, const name[], const string[]);

schreibt eine Zeichenkette in einen Registrierungsspeicherblock.

Parameter	*Erklärung*
reg	*Index des Registrierungsspeicherblocks (siehe "Indizes der Registrierungsspeicherblöcke" im Kapitel "Konstanten" auf Seite 32)*
name	*Name des Eintrags* *Wenn bereits ein Eintrag mit diesem Namen existiert, wird die bestehende Zeichenkette durch die übergebene Zeichenkette ersetzt. Andernfalls wird ein neuer Eintrag angelegt.*
string	*Array, das den zu schreibenden String enthält*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_RegSetValue(reg, const name[], {Float,Fixed,_}:value, tag=tagof value);

schreibt einen Wert in einen Registrierungsspeicherblock.

Parameter	*Erklärung*
reg	*Index des Registrierungsspeicherblocks (siehe "Indizes der Registrierungsspeicherblöcke" im Kapitel "Konstanten" auf Seite 32)*
name	*Name des Eintrags* *Wenn bereits ein Eintrag mit diesem Namen existiert, wird der bestehende Wert durch den übergebenen Wert ersetzt. Andernfalls wird ein neuer Eintrag angelegt.*
value	*zu schreibender Wert*
tag	*Anhand des "tag" des Wertes wird zwischen Integer- oder Gleitkomma-Konvertierung differenziert. - OPTIONAL*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_RegOnChg(funcidx);

legt die Funktion fest, die aufgerufen werden soll, wenn sich einer der Registrierungsspeicherblöcke geändert hat.

Parameter	*Erklärung*
funcidx	*Index der öffentlichen Funktion, die aufgerufen werden soll, wenn sich die Registrierung geändert hat* *Typ der Funktion: public func(reg);*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_SmsInit(funcidx, config);

initialisiert den SMS-Empfang

Parameter	*Erklärung*
funcidx	*Index der öffentlichen Funktion, die aufgerufen werden soll, wenn eine SMS empfangen wurde* *Typ der Funktion: public func(const SmsTel[], const SmsText[]);*
config	*reserviert für Erweiterungen*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_SmsClose();

deaktiviert den SMS-Empfang

	Erklärung
Rückgabewert	*OK*

native rM2M_WriteLog(log, param);

erzeugt einen Eintrag im Gerätelog

Parameter	*Erklärung*
log	*zu erzeugender Log-Eintrag (gültiger Bereich: 0...999).* *Die Log-Einträge werden auf den gültigen Bereich eingeschränkt. D.h. bei Unterschreiten des gültigen Bereichs wird 0 verwendet, bei Überschreiten des gültigen Bereichs wird 999 verwendet.* ***Wichtiger Hinweis:** Die Log-Einträge werden für die Anzeige am Server in den Bereich "MODULE ERR" (2000-2999) gemappt.*
param	*zusätzlicher Parameter zur genaueren Spezifizierung des Log-Eintrags (gültiger Bereich: -32768 ... 32767)*

	Erklärung
Rückgabewert	*OK*

native rM2M_SetPos(Lat, Long, Elev, Qual, SatUsed);

speichert die GPS-Positionsinformationen im Gerät. Es erfolgt keine historische Aufzeichnung. D.h. die aktuellen Positionsinformationen überschreiben immer die letzte bekannte Position. Die Informationen werden zum C-Control-Server übertragen und können z.B. per XML-Schnittstelle ausgelesen werden.

Parameter	*Erklärung*
Lat	*geographische Breite in Grad (Auflösung: 0,000001°)* *-90 000 000 = Südpol 90° Süd* *0 = Äquator* *+90 000 000 = Nordpol 90° Nord*
Long	*geographische Länge in Grad (Auflösung: 0,000001°)* *-180 000 000 = 180° West* *0 = Nullmeridian (Greenwich)* *+180 000 000 = 180° Ost*
Elev	*Höhe über dem Meeresspiegel in Meter (gültiger Bereich: -999...+9999)*
Qual	*Qualitätskennung (GPS quality indicator)* *0 = Invalid/no fix* *1 = Non-differential GPS fix* *2 = Differential GPS fix* *3 = Precise Positioning Service PPS* *4 = Real Time Kinematic RTK* *5 = Float RTK* *6 = Estimated fix (dead reckoning, Koppelnavigation)* *7 = Manual input mode* *8 = Simulation mode*
SatUsed	*Anzahl der zur Positionsbestimmung verwendeten Satelliten (gültiger Bereich: 0 ... 99)*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR* ***Hinweis:** Die Parameter werden auf die angegebenen Bereichsgrenzen geprüft. Bei Verletzung der Grenzen ist der Rückgabewert der Funktion "ERROR".*

native rM2M_SetPosNMEA(const Sentence[]);

entnimmt die GPS-Positionsinformationen aus dem übergebenen NMEA-Datensatz und speichert sie im Gerät. Es erfolgt keine historische Aufzeichnung. D.h. die aktuellen Positionsinformationen überschreiben immer die letzte bekannte Position. Die Informationen werden zum C-Control-Server übertragen und können z.B. per XML-Schnittstelle ausgelesen werden.

Parameter	*Erklärung*
Sentence	*NMEA-Datensatz vom einem GPS-Empfänger, beginnend mit dem Zeichen '$'. Derzeit werden folgende Datensätze unterstützt:* • *$GPGGA - Standortbestimmung (fix information)* ***Wichtiger Hinweis:** Die Terminierung des Strings ('\0') muss unmittelbar hinter der Checksumme erfolgen.*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "NMEA Fehlercodes" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_GetPos(&Lat, &Long, &Elev);

liest die im Gerät gespeicherten GPS-Positionsinformationen aus

Parameter	*Erklärung*
Lat	*Variable zur Aufnahme der geographischen Breite in Grad (Auflösung: 0,000001°)* *-90 000 000 = Südpol 90° Süd* *0 = Äquator* *+90 000 000 = Nordpol 90° Nord*
Long	*Variable zur Aufnahme der geographischen Länge in Grad (Auflösung: 0,000001°)* *-180 000 000 = 180° West* *0 = Nullmeridian (Greenwich)* *+180 000 000 = 180° Ost*
Elev	*Variable zur Aufnahme der Höhe über dem Meeresspiegel in Meter (gültiger Bereich: -999...+9999)*

	Erklärung
Rückgabewert	• *OK, wenn gültige GPS-Positionsinformationen im Gerät gespeichert sind* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native rM2M_GetGSMPos(posidx, pos[TrM2M_GSMPos]=0);

liefert die Anzahl der GSM-Zellen, für die gültige Informationen im Gerät gespeichert sind (posidx ‹ 0) bzw. liest die im Gerät gespeicherten Informationen über eine GSM-Zelle im Empfangsbereich aus (posidx ›= 0)

Parameter	*Erklärung*
posidx	*Auswahl der von der Funktion gelieferten Information* *posidx ‹ 0:* *Anzahl der GSM-Zellen für die gültige Informationen im Gerät gespeichert sind auslesen* *posidx ›=0:* *Nummer des GSM-Zellen-Informationsblock, der ausgelesen werden soll*
pos	*posidx ‹ 0:* *nicht erforderlich* *posidx ›=0:* *Struktur zur Aufnahme der Informationen über eine GSM-Zelle im Empfangsbereich (siehe "TrM2M_GSMPos" im Kapitel "Arrays mit symbolischen Indizes" auf Seite 30)*

	Erklärung
Rückgabewert	*posidx ‹ 0:* *Anzahl der GSM-Zellen, für die gültige Informationen im Gerät gespeichert sind (max. 10)* *posidx ›=0:* • *OK, wenn der gewünschte GSM-Zellen-Informationsblock gültige Daten enthält* • *ERROR*

native rM2M_GSMGetInfo(info[TrM2M_GSMInfo], len=sizeof info);

liefert Informationen zum GSM-Modem, SIM-Chip sowie dem bei der letzten Verbindung verwendeten GSM-Netz

Parameter	*Erklärung*
info	*Struktur zur Aufnahme der Informationen (siehe "TrM2M_GSMInfo" im Kapitel "Arrays mit symbolischen Indizes" auf Seite 30)*
len	*Größe (in Cells) der Struktur zur Aufnahme der Informationen - OPTIONAL*

	Erklärung
Rückgabewert	• *Verwendete Größe (in Cells) der Struktur zur Aufnahme der Informationen* • *ERROR, wenn Adresse und/oder Länge der Info-Struktur ungültig sind (außerhalb des Skript-Datenspeichers)* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

4.2.4 File Transfer Funktionen

4.2.4.1 Arrays mit symbolischen Indizes

TFT_Info

Eigenschaften eines Dateieintrags

```
// name       Name der Datei
// stamp      Zeitstempel der Datei (Sekunden seit 31.12.1999)
// size       Dateigröße in Byte
// crc        Ethernet CRC32 der Datei (siehe "CRC32()" )
// flags      Datei Flags (siehe "Datei Flags" im
//            Kapitel "Konstanten" auf Seite 67)
#define TFT_Info[  .name{256}, .stamp, .size, .crc, .flags ]
```

4.2.4.2 Konstanten

Datei Flags

```
FT_FLAG_READ   = 0x0001,   // Datei kann vom Server gelesen werden.
FT_FLAG_WRITE  = 0x0002,   // Datei kann vom Server geschrieben werden.
FT_FLAG_NODE   = 0x0004    /* Dateiknoten (erforderlich, um serverseitig eine
                              neue Datei anlegen zu können) */
FT_FLAG_SYSTEM = 0x0008    /* Systemdatei (kann vom Script nicht verwendet
                              werden) */
```

File Transfer Kommando

```
FT_CMD_NONE   = 0,
FT_CMD_UNLOCK = 1,         /* File Transfersitzung beendet. Der Server gibt die
                              Sperre wieder frei. */
FT_CMD_LIST   = 2,         /* Der Server fordert die Eigenschaften einer
                              Datei an */
FT_CMD_READ   = 3,         // Der Server fordert einen Block einer Datei an.
FT_CMD_STORE  = 4,         // Der Server fordert das Schreiben einer Datei an.
FT_CMD_WRITE  = 5,         /* Der Server liefert einen Block zum Schreiben in
                              eine Datei. */
FT_CMD_DELETE = 6,         // Der Server fordert das Löschen einer Datei.
```

4.2.4.3 Callback Funktionen

public func(id, cmd, const data{}, len, ofs);

vom Script-Entwickler bereitzustellende Funktion, die beim Empfang eines File Transfer Kommandos aufgerufen wird. Die Callback Funktion muss in der Lage sein, alle File Transfer Kommandos (siehe "File Transfer Kommandos" im Kapitel "Konstanten" auf Seite 67) zu behandeln.

<table>
<tr><th>Parameter</th><th>Erklärung</th></tr>
<tr><td>id</td><td>eindeutige Identifikation mit der die Datei referenziert wird (wurde bei der Registrierung festgelegt)</td></tr>
<tr><td>cmd</td><td>File Transfer Kommando, das vom System erhalten wurde und das die Callback Funktion verarbeiten muss</td></tr>
<tr><td>data</td><td>Dieser Parameter ist nur beim Empfang der folgenden File Transfer Kommandos relevant:
FT_CMD_STORE: Array, das die Eigenschaften der Datei, die neu angelegt werden soll, enthält. Aufbau:<table><tr><th>Offset</th><th>Bytes</th><th>Erklärung</th></tr><tr><td>0</td><td>4</td><td>Zeitstempel der Datei</td></tr><tr><td>8</td><td>4</td><td>Dateigröße in Byte</td></tr><tr><td>12</td><td>4</td><td>Ethernet CRC32 der Datei</td></tr><tr><td>16</td><td>2</td><td>Datei Flags</td></tr><tr><td>18</td><td>256</td><td>Name der Datei</td></tr></table>
FT_CMD_WRITE: Array, das die Daten, die vom C-Control-Server erhalten wurden, enthält.</td></tr>
<tr><td>len</td><td>Dieser Parameter ist nur beim Empfang der folgenden File Transfer Kommandos relevant:
FT_CMD_READ: Anzahl der vom C-Control-Server angeforderten Bytes
FT_CMD_STORE: Größe des vom C-Control-Server erhaltenen Dateieigenschaftenblocks
FT_CMD_WRITE: Anzahl der vom C-Control-Server erhaltenen Bytes</td></tr>
<tr><td>ofs</td><td>Dieser Parameter ist nur beim Empfang der folgenden File Transfer Kommandos relevant:
FT_CMD_READ: Byteoffset innerhalb der Datei des an den C-Control-Server zu übertragenden Datenblocks
FT_CMD_WRITE: Byteoffset innerhalb der Datei des vom C-Control-Server erhaltenen Datenblocks</td></tr>
</table>

4.2.4.4 Funktionen

native FT_Register(const name{}, id, funcidx);

registriert eine Datei, die durch das PAWN-Script zur Verfügung gestellt wird

Parameter	*Erklärung*
name	*eindeutiger Dateiname*
id	*eindeutige Identifikation mit der die Datei später referenziert wird (frei wählbar)*
funcidx	*Index der öffentlichen Funktion, die aufgerufen werden soll, wenn ein File Transfer Kommando empfangen wurde* *Typ der Funktion: public func(id, cmd, const data{}, len, ofs);* ***Wichtiger Hinweis:** Alle File Transfer Kommandos (siehe "File Transfer Kommandos" im Kapitel "Konstanten" auf Seite 67) müssen von dieser öffentlichen Funktion behandelt werden.*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native FT_Unregister(id);

entfernt eine Datei aus der Registrierung. Die Datei steht für den File Transfer nicht mehr zur Verfügung.

Parameter	*Erklärung*
id	*eindeutige Identifikation mit der die Datei referenziert wird (wurde bei der Registrierung festgelegt)*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native FT_SetProps(id, stamp, size, crc, flags);

setzt die Eigenschaften einer Datei

> ***Wichtiger Hinweis:*** *Diese Funktion muss nach dem Empfang eines "FT_CMD_LIST" Befehls aufgerufen werden.*

Parameter	*Erklärung*
id	*eindeutige Identifikation mit der die Datei referenziert wird (wurde bei der Registrierung festgelegt)*
stamp	*Zeitstempel der Datei (Sekunden seit 31.12.1999)*
size	*Dateigröße in Byte*
crc	*Ethernet CRC32 der Datei (siehe "CRC32()")*
flags	*Datei Flags (siehe "Datei Flags" im Kapitel "Konstanten" auf Seite 67)*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native FT_Read(id, const data{}, len);

übergibt die Daten an das System, um sie zum C-Control-Server zu übertragen. Bereitgestellt werden müssen die Daten durch die mittels "FT_Register()" festgelegte Callback Funktion.

> ***Wichtiger Hinweis:*** *Diese Funktion muss nach dem Empfang eines "FT_CMD_READ" Befehls aufgerufen werden.*

Parameter	*Erklärung*
id	*eindeutige Identifikation mit der die Datei referenziert wird (wurde bei der Registrierung festgelegt)*
data	*Arry, das die Daten enthält, die an das System zur Übertragung an den C-Control-Server übergeben werden sollen*
len	*Anzahl der zu übergebenden Bytes*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native FT_Accept(id, newid=-1);

akzeptiert die Datei, die der C-Control-Server schreiben will. Falls die übergebene eindeutige Identifikationsnummer (Parameter "id") auf einen Dateiknoten verweist, handelt es sich um eine neue Datei. In dem Fall muss für die neue Datei eine eindeutige Identifikationsnummer (Parameter "newid") vergeben werden. Die neue Datei muss außerdem mittels der Funktion "FT_Register()" registriert werden. Die Dateieigenschaften, die vom System an die Callback Funktion übergeben wurden (siehe "Callback Funktionen" auf Seite 68) müssen mittels der Funktion "FT_SetProps()" gespeichert werden.

> ***Wichtiger Hinweis:*** *Diese Funktion muss nach dem Empfang eines "FT_CMD_STORE" Befehls aufgerufen werden.*

Parameter	*Erklärung*
id	*eindeutige Identifikation mit der die Datei referenziert wird (wurde bei der Registrierung festgelegt)*
newid	*eindeutige Identifikation für die neue Datei, die angelegt werden soll (-1 falls es sich um keine neue Datei handelt) - OPTIONAL*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native FT_Written(id, len);

bestätigt das schreiben der Daten, die vom C-Control-Server erhalten wurden. Der eigentliche Schreibvorgang muss durch die mittels "FT_Register()" festgelegte Callback Funktion erfolgen. Der Callback Funktion werden vom System die zu schreibenden Daten (siehe "Callback Funktionen" auf Seite 68) übergeben.

> ***Wichtiger Hinweis:*** *Diese Funktion muss nach dem Empfang eines "FT_CMD_WRITE" Befehls aufgerufen werden.*

Parameter	*Erklärung*
id	*eindeutige Identifikation mit der die Datei referenziert wird (wurde bei der Registrierung festgelegt)*
len	*Anzahl der geschriebenen Bytes*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native FT_Error(id);

dient zum Anzeigen eines Fehlers im Dateihandling und beendet jeglichen Datei Befehl

Parameter	*Erklärung*
id	*eindeutige Identifikation mit der die Datei referenziert wird (wurde bei der Registrierung festgelegt)*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

4.2.5 Iotbox Funktionen

Hinweis: *Um die Funktionen dieses Kapitels verwenden zu können, benötigen Sie folgendes Include-File:* `#include <iotbox>`

4.2.5.1 Konstanten

Nummern der Universaleingänge

```
UI_CHANNEL1     = 0,            //Universaleingang 1
UI_CHANNEL2     = 1,            //Universaleingang 2

//Anzahl der Universaleingänge, über die das C-Control
                                                IoT-Starter Kit 10 verfügt
UI_NUM_CHANNELS = 2,
```

4.2.6 Universaleingangsmodul Funktionen

Hinweis: *Um die Funktionen dieses Kapitels verwenden zu können, benötigen Sie folgendes Include-File:* `#include <ui>`

4.2.6.1 Konstanten

Auswahl des Modus für einen Universaleingang

Eingangsmodi für die Funktion UI_Init()

```
UI_CHT_SI_NONE    = 0,          // deaktiviert
UI_CHT_SI_DIGITAL = 1,          // Digital
UI_CHT_SI_DCTR    = 2,          // Zähler

UI_CHT_SI_A002V   = 6,          // 0...2,5V
```

Samplerate in [Hz] für die Messung

```
UI_SAMPLE_RATE_2   =   2,
UI_SAMPLE_RATE_4   =   4,
UI_SAMPLE_RATE_8   =   8,
UI_SAMPLE_RATE_16  =  16,
UI_SAMPLE_RATE_32  =  32,
UI_SAMPLE_RATE_64  =  64,
UI_SAMPLE_RATE_128 = 128,
```

4.2.6.2 Funktionen

native UI_Init(channel, mode, filtertime);

initialisiert einen Universaleingang (UI 1 - UI 2). Die Konfiguration der Samplerate für die Messwerterfassung erfolgt durch die Funktion "UI_SetSampleRate". Ein Aufruf der Funktion "UI_SetSampleRate" ist nur erforderlich, wenn die Default-Einstellung der Samplerate von 16Hz (62,5ms) für Ihre Anwendung nicht geeignet ist.

Hinweis: *Mit jedem Universaleingang der initialisiert wird, steigt der Energieverbrauch.*

Parameter	*Erklärung*
channel	*Nummer des Universaleingangs, beginnend mit 0 für UI 1* ***Hinweis:*** *Sie können für diesen Parameter auch die vordefinierten Konstanten verwenden (siehe "Nummern der Universaleingänge" im Kapitel "Iotbox Funktionen" auf Seite 72).*
mode	*Auswahl des Modus für den Universaleingang* *UI_CHT_SI_NONE : Universaleingang deaktiviert* *UI_CHT_SI_DIGITAL : Digital: max. 2,8V, low ‹0,84V, high ›1,96V, Bürde 1MΩ* *UI_CHT_SI_DCTR : Zähler: Impulslänge min. 1ms, Bürde 1MΩ* *UI_CHT_SI_A002V : 0...2,5V: Auflösung 610µV, max. 2,5V, Bürde 1MΩ*
filtertime	*Modi "Digital" und "Zähler":* *Zeit in [ms], für die ein Signal konstant anliegen muss, um einen Pegelwechsel auszulösen. Dient zur Unterdrückung von kurzzeitigen Störungen (Entprellung).* *Modus "0...2,5V":* *Zeit in [ms], über die das Analogsignal zwecks Signalglättung gemittelt wird. Dient zur Unterdrückung von Signalrauschen.*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *‹ OK, wenn ein Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

native UI_Close(channel);

deaktiviert einen Universaleingang (UI 1 - UI 2).

Hinweis: *Mit jedem Universaleingang der deaktiviert wird, sinkt der Energieverbrauch.*

Parameter	*Erklärung*
channel	*Nummer des Universaleingangs, beginnend mit 0 für UI 1* ***Hinweis:*** *Sie können für diesen Parameter auch die vordefinierten Konstanten verwenden (siehe "Nummern der Universaleingänge" im Kapitel "Iotbox Funktionen" auf Seite 72).*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR, wenn ein ungültiger Parameter übergeben wurde*

native UI_GetValue(channel, &value=0);

liest den letzt gültigen Messwert für den angegebenen Universaleingang vom System.

Parameter	*Erklärung*
temp	*Nummer des Universaleingangs, beginnend mit 0 für UI 1* ***Hinweis:*** *Sie können für diesen Parameter auch die vordefinierten Konstanten verwenden (siehe "Nummern der Universaleingänge" im Kapitel "Iotbox Funktionen" auf Seite 72).*
value	*Variable zur Aufnahme des zu lesenden Messwerts. Wie der gelesene Messwert zu interpretieren ist, ist vom Modus des Universaleingangs abhängig.* *UI_CHT_SI_NONE : ---* *UI_CHT_SI_DIGITAL : Digital: 0 =^ "low", 1 =^ "high"* *UI_CHT_SI_DCTR : Zählerstand []* *UI_CHT_SI_A002V : 0...2V: Spannung in [mV]*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR, wenn ein ungültiger Parameter übergeben wurde*

native UI_SetSampleRate(samplerate);

setzt die Samplerate für die Messwerterfassung an den Universaleingängen. Die getroffene Einstellung ist immer für alle Universaleingänge gültig. Eine gesonderte Einstellung für einzelne Universaleingänge ist nicht möglich. Der Default-Wert der Samplerate beträgt 16Hz (62,5ms) .

Hinweis: *Mit Erhöhung der Samplerate steigt auch der Energieverbrauch.*

Parameter	*Erklärung*
samplerate	*Sampelrate in [Hz]* ***Hinweis:*** *Für diesen Parameter sind nur die unter "Samplerate in [Hz] für die Messung" angegebenen Konstanten im Kapitel "Konstanten" auf Seite 72 zulässig.*

	Erklärung
Rückgabewert	• *OK, wenn erfolgreich* • *ERROR, wenn ein ungültiger Parameter übergeben wurde*

Hinweis: *Für Universaleingänge, die im Modus "Zähler" betrieben werden, ist die Samplerate nicht von Bedeutung. Sollten Sie alle Universaleingänge in diesem Modus betreiben, können Sie für die Samplerate den niedrigst möglichen Wert verwenden.*

native UI_ResetCounter(channel);

setzt den Zählerstand eines Universaleingangs, der im Modus "Zähler" betrieben wird, zurück. Trat dabei kein Fehler auf, liefert die Funktion als Rückgabewert den Zählerstand vor dem Zurücksetzen des Zählers.

Parameter	*Erklärung*
channel	*Nummer des Universaleingangs, beginnend mit 0 für UI 1* ***Hinweis:*** *Sie können für diesen Parameter auch die vordefinierten Konstanten verwenden (siehe "Nummern der Universaleingänge" im Kapitel "Iotbox Funktionen" auf Seite 72).*

	Erklärung
Rückgabewert	• *Zählerstand vor dem Zurücksetzen* • *ERROR, wenn ein ungültiger Parameter übergeben wurde*

4.2.7 Mathematische Funktionen

Hilfreiche Konstanten

Definintition	Wert	Beschreibung
M_E	2.7182818284590452354	e
M_LOG2E	1.4426950408889634074	$\log_2 e$
M_LOG10E	0.43429448190325182765	$\log_{10} e$
M_LN2	0.69314718055994530942	ln 2
M_LN10	2.30258509299404568402	ln 10
M_PI	3.14159265358979323846	π
M_PI_2	1.57079632679489661923	π/2
M_PI_4	0.78539816339744830962	π/4
M_1_PI	0.31830988618379067154	1/π
M_2_PI	0.63661977236758134308	2/π
M_2_SQRTPI	1.12837916709551257390	2/sqrt(π)
M_SQRT2	1.41421356237309504880	sqrt(2)
M_SQRT1_2	0.70710678118654752440	1/sqrt(2)

native fround(Float:x);

führt kaufmännisches Runden des übergebenen Floats durch

Parameter	*Erklärung*
x	*Float, der gerundet werden soll*

	Erklärung
Rückgabewert	*kaufmännisch gerundeter ganzzahliger Wert*

Die Arbeitsweise der folgenden Funktionen entspricht jener der Standard ANSI-C Implementierung:

native Float:sin(Float:x);

Sinus von x

native Float:cos(Float:x);

Kosinus von x

native Float:tan(Float:x);

Tangens von x

native Float:asin(Float:x);

arcsin(x) im Bereich [-π/2, π/2], x Element von [-1, 1]

native Float:acos(Float:x);
arccos(x) im Bereich [0, π], x Element von [-1, 1]

native Float:atan(Float:x);
arctan(x) im Bereich [-π/2, π/2]

native Float:atan2(Float:y, Float:x);
arctan(y/x) im Bereich [-π, π]

native Float:sinh(Float:x);
Sinus Hyperbolicus von x

native Float:cosh(Float:x);
Cosinus Hyperbolicus von x

native Float:tanh(Float:x);
Tangens Hyperbolicus von x

native Float:exp(Float:x);
Exponentialfunktion e^x

native Float:log(Float:x);
natürlicher Logarithmus ln(x), x › 0

native Float:log10(Float:x);
Logarithmus zur Basis 10 $\log_{10}(x)$, x › 0

native Float:pow(Float:x, Float:y);
x^y. Ein Argumentenfehler liegt vor bei x = 0 und y ‹= 0, oder bei x ‹ 0 und y ist nicht ganzzahlig.

native Float:sqrt(Float:x);
Wurzel x, x ›= 0

native Float:ceil(Float:x);
kleinster ganzzahliger Wert, der nicht kleiner als x ist

native Float:floor(Float:x);
größter ganzzahliger Wert, der nicht größer als x ist

native Float:fabs(Float:x);
absoluter Wert | x |

native Float:ldexp(Float:x, n);
$x*2^n$

native Float:frexp(Float:x, &n);
zerlegt x in eine normalisierte Mantisse im Bereich [1/2, 1], die als Resultat geliefert wird, und eine Potenz von 2, die in n abgelegt wird. Ist x null, sind beide Teile des Resultats null.

native Float:modf(Float:x, &Float:ip);
zerlegt x in einen ganzzahligen Teil und einen Rest, die beide das gleiche Vorzeichen wie x besitzen. Der ganzzahlige Teil wird bei ip abgelegt, der Rest ist das Resultat.

native Float:fmod(Float:x, Float:y);
Gleitpunktrest von x/y, mit dem gleichen Vorzeichen wie x. Wenn y null ist, hängt das Resultat von der Implementierung ab.

native isnan(Float:x);
liefert einen Wert ungleich Null, wenn x "not a number" ist

4.2.8 String Funktionen

> ***Hinweis:*** *Um die Funktionen dieses Kapitels verwenden zu können, benötigen Sie folgendes Include-File:* `#include <string>`

Die Arbeitsweise der folgenden Funktionen entspricht im Wesentlichen jener der Standard ANSI-C Implementierung:

native strlen(const string[]);

liefert die Länge von string (ohne '\o')

Parameter	*Erklärung*
string	*Zeichenkette, deren Länge bestimmt werden soll*

	Erklärung
Rückgabewert	*Anzahl der Zeichen ohne der abschließenden '\o'*

native sprintf(dest[], maxlength=sizeof dest, const format[], {Float,Fixed,_}:...);

speichert den übergebenen Format-String in dem Array dest. Die Arbeitsweise der Funktionen entspricht der Funktion "snprintf" der Standard ANSI-C Implementierung

Parameter	*Erklärung*
dest	*Array zur Aufnahme des formatierten Ergebnisses*
maxlength	*maximale Zeichenanzahl, die das dest Array aufnehmen kann*
format	*die zu verwendende Format-Zeichenkette*

	Erklärung
Rückgabewert	• *-1 im Fehlerfall* • *Anzahl der Zeichen, die geschrieben worden wäre, wenn das Array dest lang genug gewesen wäre (ohne '\o').* *Das Array dest erhält in jedem Fall ein abschließendes Nullzeichen. In keinem Fall wird über die Länge des Arrays dest hinausgeschrieben.*

native strcpy(dest[], const source[], maxlength=sizeof dest);

kopiert die Zeichenkette source in das Array dest (inklusive '\o').

Parameter	*Erklärung*
dest	*Array zur Aufnahme der zu kopierenden Zeichenkette*
source	*zu kopierende Zeichenkette*
maxlength	*Anzahl der zu kopierenden Zeichen - OPTIONAL*

	Erklärung
Rückgabewert	*Anzahl der kopierten Zeichen*

native strcat(dest[], const source[], maxlength=sizeof dest);

fügt die Zeichenkette source an die Zeichenkette dest an (inklusive '\o')

Parameter	*Erklärung*
dest	*Array zur Aufnahme des Ergebnisses. Dieses Array enthält bereits eine Zeichenkette an die die Zeichenkette source angefügt werden soll.*
source	*Zeichenkette, die an die im Array dest enthaltene Zeichenkette angefügt werden soll*
maxlength	*Anzahl der anzufügenden Zeichen - OPTIONAL*

	Erklärung
Rückgabewert	*Anzahl der angefügten Zeichen*

native strcmp(const string1[], const string2[], length=cellmax);

vergleicht die Zeichenketten string1 und string2

Parameter	*Erklärung*
string1	*die beiden Zeichenketten, die verglichen werden sollen*
string2	
length	*Die maximale Anzahl von Zeichen, die beim Vergleich berücksichtigt werden sollen - OPTIONAL*

	Erklärung
Rückgabewert	• *1: string1 › string 2* • *0: die beiden Zeichenketten sind gleich (zumindest die berücksichtigte Länge)* • *-1: string1 ‹ string 2*

native strchr(const string[], char);

sucht ein Zeichen (erstes Vorkommen) in einer Zeichenkette

Parameter	*Erklärung*
string	*Zeichenkette, die durchsucht werden soll*
char	*Zeichen, das gesucht werden soll*

	Erklärung
Rückgabewert	• *-1, wenn das gesuchte Zeichen nicht in der Zeichenkette enthalten ist* • *Array-Index des gesuchten Zeichens (erstes in der Zeichenkette vorkommendes Zeichen)*

native strrchr(const string[], char);

sucht ein Zeichen (letztes vorkommen) in einer Zeichenkette

Parameter	*Erklärung*
string	*Zeichenkette, die durchsucht werden soll*
char	*Zeichen, das gesucht werden soll*

	Erklärung
Rückgabewert	• *-1, wenn das gesuchte Zeichen nicht in der Zeichenkette enthalten ist* • *Array-Index des gesuchten Zeichens (letztes in der Zeichenkette vorkommendes Zeichen)*

native strspn(const string1[], const string2[]);

sucht die Position des ersten Zeichens in string1, das ***nicht*** *in der Zeichenkette erlaubter Zeichen (string2) enthalten ist*

Parameter	*Erklärung*
string1	*Zeichenkette, die durchsucht werden soll*
string2	*Zeichenkette erlaubter Zeichen*

	Erklärung
Rückgabewert	• *Länge von string1, wenn keine unerlaubten Zeichen gefunden wurden* • *Position des ersten Zeichens in der zu durchsuchenden Zeichenkette, das nicht in der Zeichenkette der erlaubten Zeichen enthalten ist*

native strcspn(const string1[], const string2[]);

sucht die Position des ersten Zeichens in string1, das auch in der Zeichenkette erlaubter Zeichen (string2) enthalten ist

Parameter	*Erklärung*
string1	*Zeichenkette, die durchsucht werden soll*
string2	*Zeichenkette erlaubter Zeichen*

	Erklärung
Rückgabewert	• *Länge von string1, wenn kein erlaubtes Zeichen gefunden wurde* • *Position des ersten Zeichens in der zu durchsuchenden Zeichenkette, das auch in der Zeichenkette der erlaubten Zeichen enthalten ist*

native strpbrk(const string1[], const string2[]);

sucht den Array-Index des ersten Zeichens, das auch in der Zeichenkette erlaubter Zeichen enthalten ist

Parameter	*Erklärung*
string1	*Zeichenkette, die durchsucht werden soll*
string2	*Zeichenkette erlaubter Zeichen*

	Erklärung
Rückgabewert	• *-1, wenn das gesuchte Zeichen nicht in der Zeichenkette enthalten ist* • *Array-Index des ersten Zeichens in der zu durchsuchenden Zeichenkette, das auch in der Zeichenkette der erlaubten Zeichen enthalten ist*

native strstr(const string1[], const string2[]);

sucht die Zeichenkette string2 in der Zeichenkette string1

Parameter	*Erklärung*
string1	*Zeichenkette, die durchsucht werden soll*
string2	*zu suchende Zeichenkette*

	Erklärung
Rückgabewert	• *-1, wenn die zu suchende Zeichenkette string2 nicht in string1 enthalten ist* • *Array-Index an der die zu suchende Zeichenkette string2 im string1 beginnt*

native strtol(const string[], base);

wandelt eine Zeichenkette in einen Wert um

Parameter	*Erklärung*
string	*umzuwandelnde Zeichenkette*
base	*gibt die Basis an, die für die Umwandlung verwendet werden soll* *2-36: Die angegebene Basis wird verwendet* *0: Als Basis wird 8, 10 oder 16 verwendet, abhängig von der umzuwandelnden Zeichenkette* *Basis 8: bei einer führenden 0* *Basis 16: bei 0x oder 0X*

	Erklärung
Rückgabewert	*Wert, der der Zeichenkette entspricht*

native Float: atof(const string[]);

wandelt eine Zeichenkette in einen Float um

Parameter	*Erklärung*
string	*umzuwandelnde Zeichenkette*

	Erklärung
Rückgabewert	*Float, dessen Zahlenwert der Zeichenkette entspricht*

4.2.9 Hilfsfunktionen

4.2.9.1 Arrays mit symbolischen Indizes

TablePoint

zweispaltige Stützpunkttabelle, Datentyp Integer

```
// key     Spalte, die durchsucht wird
// value   Spalte mit den zurückzuliefernden Ergebniswerten

#define TablePoint[.key, .value]
```

TablePointF

zweispaltige Stützpunkttabelle, Datentyp Float

```
// key     Spalte, die durchsucht wird
// value   Spalte mit den zurückzuliefernden Ergebniswerten

#define TablePointF[Float:.key, Float:.value]
```

4.2.9.2 Konstanten

Fehlercodes der Funktionen "CalcTable" und "CalcTableF"

```
const
{
  TAB_ERR_FLOOR = -1, // gesuchter Wert kleiner als der erste Tabelleneintrag
  TAB_ERR_CEIL = -2,  // gesuchter Wert größer als der letzte Tabelleneintrag
};
```

4.2.9.3 Funktionen

native getapilevel();

gibt das implementierte API-Level der Skript-Engine aus

	Erklärung
Rückgabewert	*implementiertes API-Level der Skript-Engine*

native CRC16(data{}, len);

liefert die berechnete Modbus CRC16 der übergebenen Daten

Parameter	*Erklärung*
data	*Array, das die Daten enthält für die die CRC16 berechnet werden soll*
len	*Anzahl der Bytes, die bei der Berechnung berücksichtigt werden sollen*

	Erklärung
Rückgabewert	*berechnete CRC16*

native CRC32(data{}, len);

liefert die berechnete Ethernet CRC32 der übergebenen Daten

Parameter	*Erklärung*
data	*Array, das die Daten enthält für die die CRC32 berechnet werden soll*
len	*Anzahl der Bytes, die bei der Berechnung berücksichtigt werden sollen*

	Erklärung
Rückgabewert	*berechnete CRC32*

native CalcTable(key, &value, const table[][TablePoint], size = sizeof table);

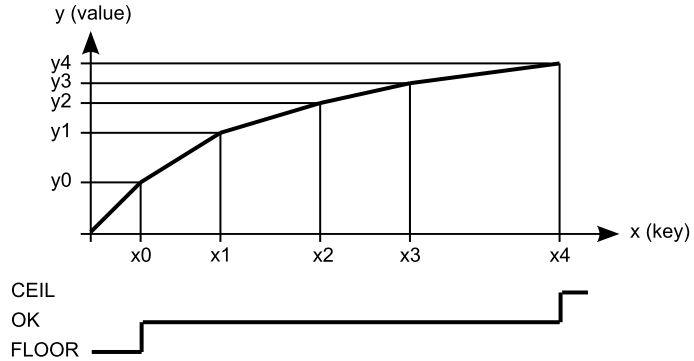
*sucht einen bestimmten Wert in der "key"-Spalte der übergebenen Stützpunktabelle und liefert den entsprechenden Wert der "value"-Spalte der Tabelle. Liegt der gesuchte Wert zwischen zwei Stützpunkten, wird der Rückgabewert zwischen den zwei angrenzenden "value"-Spaltenwerten linear interpoliert (Geradengleichung: y = k*x + d). Mit dieser Funktion können nicht lineare Kennlinien (z.B. Zusammenhang ADC-Wert -› Temperatur) nachgebildet werden.*

Parameter	*Erklärung*
key	*Wert, der für die Suche herangezogen wird*
value	*enthält das Ergebnis der Berechnung durch die Funktion*
table	*Die Tabelle, die durchsucht wird, muss vom Typ "TablePoint" sein.*
size	*Anzahl der Zeilen der Tabelle*

	Erklärung
Rückgabewert	• *OK, wenn der entsprechende Wert gefunden wurde* • *TAB_ERR_FLOOR, wenn der gesuchte Wert kleiner als der erste Tabelleneintrag ist. "value" beinhaltet den ersten Tabelleneintrag.* • *TAB_ERR_CEIL, wenn der gesuchte Wert größer als der letzte Tabelleneintrag ist. "value" beinhaltet den letzten Tabelleneintrag.* • *‹ OK, wenn ein anderer Fehler auftritt (siehe "Returncodes für allgemeine Zwecke" im Kapitel "Konstanten" auf Seite 32)*

Hinweis: *Ergänzende Erklärung zur Stützpunkttabelle* `"table"`

Die Tabellenzeilen können in einem x/y-Koordinatensystem dargestellt werden. Die Werte der "key"-Spalte werden dabei auf der x-Achse aufgetragen, die dazugehörenden Werte der "value"-Spalte auf der y-Achse.



Darstellung der Stützpunkttabelle als x/y-Koordinatensystem

native CalcTableF(Float:key, &Float:value, const table[][TablePointF], size = sizeof table);

die Funktionsweise entspricht der "CalcTable" Funktion. Der Unterschied besteht darin, dass "Float" der Datentyp für alle Elemente der "CalcTableF" Funktion ist.

4.3 Pawn Script Fehlercodes

Sollte beim Durchlaufen des Pawn Scripts ein Fehler auftreten, wird die Scriptausführung gestoppt und deaktiviert. Zudem wird in das Gerätelog der entsprechende Fehlercode eingetragen. Bei allen Log-Einträgen bis auf "SCRIPT_ERR" enthält der Parameter den 32-Bit Instruction Pointer der PAWN abstract machine (AMX). Da im Parameter eines Log-Eintrags nur 16-Bit Werte gespeichert werden können, werden jeweils 2 Einträge im Gerätelog erzeugt. Der erste Eintrag enthält Bit31-Bit16 und der zweite Eintrag enthält Bit15-Bito des 32-Bit Instruction Pointers. Eine Anleitung zum Auswerten des Gerätelogs finden Sie im Kapitel "Kateireiter "Log""(siehe "Karteireiter "Log"" auf Seite 22).

Log-Eintrag		Parameter		Beschreibung
Code	Klartext	Code	Klartext	
3000	SCRIPT_ERR	0	NO SCRIPT	kein gültiges Script vorhanden
		1	SCRIPT UPDATE	neues Script erhalten
		2	SCRIPT EXCEPT LOOP	Exception Loop erkannt (4 Systemstarts nach Exeption innerhalb von 10min.) Das Script wird deaktiviert und das Errorhandling aktiviert (siehe "Errorhandling" auf Seite 90). Es erfolgt auch eine Neu-Formatierung des Filesystems. D.h. alle bisher aufgezeichneten Daten sowie Log-Einträge gehen verloren. Dies wird durch den zusätzlichen Log-Eintrag "LOG REFORMATFILE" signalisiert.
		3	SCRIPT SOFT ERROR 1	Erstmaliges Auftreten eines Laufzeitfehlers innerhalb von 24h. Script wurde neu gestartet.
		4	SCRIPT SOFT ERROR 2	Laufzeitfehler zum zweiten Mal innerhalb von 24h aufgetreten. Script wurde neu gestartet.
		5	SCRIPT SOFT ERROR 3	Laufzeitfehler zum dritten Mal innerhalb von 24h aufgetreten. Script wurde neu gestartet. Sollte innerhalb von 24h ein weiterer Laufzeitfehler auftreten, wird das Script deaktiviert und das Errorhandling aktiviert (siehe "Errorhandling" auf Seite 90).
		6	SCRIPT UPDATE ERROR	Verbindungsabbruch während des Scriptdownloads Das bestehende Script kann dadurch, dass es sich im RAM befindet, bis zum nächsten PowerOn weiterhin ausgeführt werden. Nach einem PowerOn ist die Ausführung nicht mehr möglich und es wird der Fehler "SCRIPT_ERR, NO SCRIPT" ins Gerätelog eingetragen.
3001	AMX_ERR_EXIT	##	---	Abbruch z.B. Max. Anzahl der PAWN-Befehle (100.000) pro Durchlauf erreicht
3002	AMX_ERR_ASSERT	##	---	Assertion fehlgeschlagen

Log-Eintrag		Parameter		Beschreibung
Code	Klartext	Code	Klartext	
3003	AMX_ERR_ STACKERR	##	---	Stack / Heap Kollision (unzureichende Stack-Größe)
3004	AMX_ERR_BOUNDS	##	---	Array-Index außerhalb des gültigen Bereichs
3005	AMX_ERR_ MEMACCESS	##	---	ungültiger Speicherzugriff z.B. Verwechslung zwischen Cell (32Bit Element) Zugriff [] und Byte Zugriff {}
3006	AMX_ERR_INVINSTR	##	---	ungültige Anweisung
3007	AMX_ERR_ STACKLOW	##	---	Stack-Unterlauf
3008	AMX_ERR_ HEAPLOW	##	---	Heap Unterlauf
3009	AMX_ERR_ CALLBACK	##	---	keine (ungültige) native Callback-Funktion
3010	AMX_ERR_NATIVE	##	---	Native-Funktion ist fehlgeschlagen
3011	AMX_ERR_DIVIDE	##	---	Division durch Null
3012	AMX_ERR_SLEEP	##	---	Sleep-Modus
3013	AMX_ERR_ INVSTATE	##	---	ungültiger Zustand
3014	reserviert			
3015	reserviert			
3016	AMX_ERR_MEMORY	##	---	out of memory
3017	AMX_ERR_FORMAT	##	---	ungültiges/nicht unterstütztes P-Code Dateiformat
3018	AMX_ERR_VERSION	##	---	Datei ist für eine neuere Version des AMX
3019	AMX_ERR_ NOTFOUND	##	---	Datei oder Funktion nicht gefunden
3020	AMX_ERR_INDEX	##	---	ungültiger Index-Parameter (ungültiger Einstiegspunkt)
3021	AMX_ERR_DEBUG	##	---	Debugger kann nicht ausgeführt werden
3022	AMX_ERR_INIT	##	---	AMX nicht initialisiert (oder doppelt initialisiert)
3023	AMX_ERR_ USERDATA	##	---	Benutzer-Datenfeld kann nicht gesetzt werden (Tabelle voll)
3024	AMX_ERR_INIT_JIT	##	---	JIT kann nicht initialisiert werden.

Log-Eintrag		Parameter		Beschreibung
Code	Klartext	Code	Klartext	
3025	AMX_ERR_PARAMS	##	---	fehlerhafter Parameter
3026	AMX_ERR_DOMAIN	##	---	Domain-Fehler. Das Ergebnis des Ausdrucks ist nicht im gültigen Bereich.
3027	AMX_ERR_GENERAL	##	---	allgemeiner Fehler (unbekannter oder nicht spezifizierter Fehler)
3028	AMX_ERR_OVERLAY	##	---	Overlays werden nicht unterstützt (JIT) oder sind nicht initialisiert.

4.3.1 Errorhandling

Um zu gewährleisten, dass bei Problemen mit dem Script eine Diagnose bzw. Behebung aus der Ferne möglich ist, wurden folgende Übertragungmechanismen in die Firmware integriert. Für den Fall, dass kein Script vorhanden ist, erfolgt die Verbindung zum C-Control-Server alle 24h . Sollte ein vorhandenes Script aufgrund von vom System erkannten Fehlern deaktiviert worden sein, wird dieses Backup Intervall auf 1h gesetzt. In beiden Fällen wird die Verbindugsart „Intervall & Wakeup“ aktiviert, wodurch das Auslösen einer Verbindung über die Oberfläche des C-Control-Servers möglich ist (siehe "Benutzerhandbuch für C-Control -Server " 206.886).

4.4 Syntax

4.4.1 Allgemeine Syntax

4.4.1.1 Format

Bezeichner, Zahlen und Zeichen werden durch Leerzeichen, Tabulatoren, Zeilenumbrüche und "Form Feed" getrennt. Eine Serie von einer oder mehreren dieser Separatoren wird als Leerraum erkannt.

4.4.1.2 Optionale Semikolons

Semikolons (um ein Statement zu beenden) sind optional, wenn sie am Ende einer Zeile auftreten. Semikolons sind notwendig, um mehrere Statements in einer Zeile zu trennen. Ein Ausdruck kann auf mehrere Zeilen aufgeteilt werden, jedoch müssen Postfix-Operatoren in derselben Zeile wie der Operand stehen.

4.4.1.3 Kommentare

Text zwischen den Symbolen /* und */ (beide Symbole können auf derselben oder auf unterschiedlichen Zeilen stehen) und Text nach // (bis zum Ende einer Zeile) sind Kommentare. Kommentare dürfen nicht verschachtelt werden. Der Compiler betrachtet Kommentare als Leerzeichen. Ein Kommentar, der mit "/** " (zwei Sterne und ein Leerzeichen nach dem zweiten

Stern) beginnt und mit einem "*/" endet, ist ein Dokumentationskommentar. Ein Kommentar, der mit "/// " (drei Schrägstriche und ein Leerzeichen nach dem dritten Schrägstrich) beginnt, ist ebenfalls ein Dokumentationskommentar. Der Parser kann den Dokumentationskommentar in unterschiedlicher Weise unterstützen, zum Beispiel könnte er eine Online-Hilfe daraus generieren.

4.4.1.4 Bezeichner

Namen von Variablen, Funktionen und Konstanten. Bezeichner bestehen aus den Zeichen a...z, A...Z, 0...9, _ oder @. Das erste Zeichen darf keine Ziffer sein. Die Zeichen @ und _ alleine sind keine gültigen Bezeichner, z.B. "_Up" ist ein gültiger Bezeichner, aber "_" ist es nicht. Pawn unterscheidet zwischen Groß- und Kleinschreibung. Der Parser schneidet Bezeichner ab einer bestimmten Länge ab. Es werden standardmäßig nur die ersten 16 Zeichen für die Unterscheidung herangezogen.

4.4.1.5 Reservierte Schlüsselworte

Statements	Operator	Direktiven	Andere
assert	defined	defined	defined
break	sizeof	sizeof	sizeof
case	state	state	state
continue	tagof	tagof	tagof
default			
do			
else			
exit			
for			
goto			
if			
return			
sleep			
state			
switch			
while			

4.4.1.6 Numerische Konstanten

4.4.1.6.1 Numerische Integer-Konstanten

Binär
ob gefolgt von einer Serie von 0 und 1

Dezimal:
eine Serie von Ziffern zwischen 0 und 9

Hexadezimal
ox gefolgt von einer Serie von Ziffern zwischen 0 und 9 und den Buchstaben a bis f

4.4.1.6.2 Numerische Gleitkomma-Konstanten

Eine Gleitkommazahl ist eine Zahl mit einem Nachkommateil. Eine Gleitkommazahl beginnt mit einer oder mehreren Ziffern, beinhaltet einen Dezimaltrennpunkt und hat zumindest eine Ziffer nach dem Dezimaltrennpunkt. z.B. "12.0" und "0.75" sind gültige Gleitkommazahlen. Optional kann noch ein Exponent angehängt werden. Die Notation ist der Buchstabe "e" (Kleinbuchstabe), gefolgt von einer ganzzahligen numerischen Konstante. Z.B. "3.12e4" oder "12.3e-3" sind gültige Gleitkommazahlen mit Exponent.

4.4.2 Variablen

4.4.2.1 Deklaration

Das Schlüsselwort "new" deklariert eine neue Variable. Für spezielle Deklarationen wird das Schlüsselwort "new" durch "static" ersetzt (siehe "Statische lokale Deklaration" auf Seite 92). Sofern sie nicht explizit initialisiert wird, ist der Wert der neuen Variablen Null.

Eine Variablendeklaration kann auftreten

- an jeder Position, an der ein Ausdruck gültig ist - lokale Variable
- an jeder Position, an der eine Funktionsdeklaration oder eine Implementation der Funktion gültig ist - globale Variablen;
- im ersten Ausdruck einer "for" Schleife (siehe "for (Ausdruck 1 ; Ausdruck 2 ; Ausdruck 3) Statement " auf Seite 104) - lokale Variable

Beispiel:

```
new a;     // ohne Initialisierung (Wert ist 0)
new b = 3; // mit Initialisierung (Wert ist 3)
```

4.4.2.2 Lokale Deklaration

Eine lokale Deklaration erscheint innerhalb eines Anweisungs-Blocks. Auf eine Variable kann nur innerhalb dieses Blocks und der darin enthaltenen Blöcke zugegriffen werden. Eine Deklaration innerhalb des ersten Ausdrucks einer Schleifenanweisung ist ebenfalls eine lokale Deklaration.

4.4.2.3 Globale Deklaration

Eine globale Deklaration erscheint außerhalb einer Funktion und eine globale Variable kann in jeder Funktion verwendet werden. Globale Variablen können nur mit konstanten Ausdrücken initialisiert werden.

4.4.2.4 Statische lokale Deklaration

Eine lokale Variable wird zerstört, wenn die Ausführung den Block verlässt, in dem die Variable geschaffen wurde. Lokalen Variablen in einer Funktion existieren nur während der Laufzeit der genannten Funktion. Jede neuer Aufruf der Funktion erstellt und initialisiert neue lokale Variablen. Wenn eine lokale Variable mit dem Schlüsselwort "static" anstatt "new" deklariert ist, bleibt die

Variable auch nach dem Ende einer Funktion im Speicher. Dies bedeutet, dass statische lokale Variablen eine private, dauerhafte Speicherung bereitstellen, die nur in einer einzigen Funktion (oder einem Block) zugänglich sind. Wie globale Variablen, können statische lokale Variablen nur mit konstanten Ausdrücken initialisiert werden.

4.4.2.5 Statische globale Deklaration

Eine statische globale Variable verhält sich wie eine globale Variable, mit dem Unterschied, dass die Variable nur in der Datei gültig ist, in der sie deklariert wurde. Um eine globale Variable statisch zu deklarieren, ersetzen Sie das Schlüsselwort "new" mit "static".

4.4.2.6 Gleitkommawerte

Pawn unterstützt Gleitkommawerte. Diese können an jeder Stelle eingesetzt werden, an der eine Variablendeklaration gültig ist.

Beispiel:

```
new Float:a;       // ohne Initialisierung (Wert ist 0.0)
new Float:b = 3.0; // mit Initialisierung (Wert ist 3.0)
```

4.4.3 Konstante Variablen

Es ist manchmal notwendig eine Variable zu erstellen, die einmal initialisiert wird und dann nicht mehr verändert werden soll. Eine solche Variable verhält sich ähnlich wie eine symbolische Konstante, aber sie ist dennoch eine Variable. Um eine konstante Variable zu deklarieren, legen Sie das Schlüsselwort "const" zwischen das Schlüsselwort, das die Variablendeklaration ("new", "static") startet und den Namen der Variablen.

Beispiel:

```
new const address[4] = { 192, 0, 168, 66 }
static const status /* initialized to zero */
```

Typische Situationen, in denen man eine konstante Variable nutzen könnte, sind:

- Um eine "array"-Konstante zu erstellen. Auf symbolische Konstanten kann nicht per Index zugegriffen werden.
- Ein besonderer Fall ist, wenn die Array-Argumente in einer Funktion als "const" markiert werden. Array-Argumente werden immer per Referenz übergeben. Wenn sie als "const" deklariert werden, schützt sie das vor ungewollten Änderungen. Siehe Beispiele von "const-Funktionsargumenten" im Kapitel "Funktionsargumente ("call-by-value" versus "call-by-reference")" auf Seite 109.

4.4.4 Array Variablen

4.4.4.1 Eindimensionales Array

Die Syntax name[constant] deklariert "name" als ein Array aus “constant” Elementen, wobei jedes Element ein Eintrag ist. "name" ist ein Platzhalter für den Namen der Variable und "constant" ist ein positiver Wert ungleich Null. "constant" ist optional und kann weggelassen werden. Wenn kein Wert zwischen den Klammern steht, ist die Anzahl von Elementen gleich der Anzahl der Initialwerte. Der Array-Index-Bereich ist "Null-basierend", das bedeutet, dass das erste Element "name[0]" und das letzte Element "name[constant-1]" ist.

4.4.4.2 Initialisierung

Datenobjekte können bei ihrer Deklaration initialisiert werden. Der initialisierte Wert von globalen Datenobjekten muss ein konstanter Wert sein. Arrays, global oder lokal, müssen ebenfalls mit konstanten Werten initialisiert werden. Nicht initialisierte Daten sind standardmäßig Null.

Beispiele:

Auflistung: gültige Deklaration

```
new i = 1
new j                           /* j ist 0 */
new k = 'a'                     /* k hat den Zeichencode von 'a' */
new a[] = [1,4,9,16,25]         /* a hat 5 Elemente */
new s1[20] = ['a','b']          /* die restlichen 18 Elemente sind 0 */
new s2[] = ''Hello world...''   /* ein unpacked string */
```

Auflistung: ungültige Deklaration

```
new c[3] = 4                    /* Ein Array kann nicht auf einen einzelnen
                                   Wert gesetzt werden */
new i = "Good-bye"              /* Nur ein Array kann einen String halten. */
new q[]                         /* Unbekannte Größe für ein Array */
new p[2] = { i + j, k - 3 }     /* Arrayinitialisierer müssen Konstanten sein. */
```

4.4.4.3 Progressive Initialisierung für Arrays

Der Punkte-Operator führt die Initialisierung des Arrays aufgrund der letzten beiden initialisierten Werte weiter. Der Punkte-Operator (drei Punkte, "...") initialisiert das Array bis zur Arraygrenze.

Beispiel: Auflistung: Arrayinitialisierer

```
new a[10] = { 1, ... }          // setzt alle Elemente auf 1
new b[10] = { 1, 2, ... }       // b = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
new c[8]  = { 1, 2, 40, 50, ... } // c = 1, 2, 40, 50, 60, 70, 80, 90
new d[10] = { 10, 9, ... }      // d = 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
```

4.4.4.4 Mehrdimensionale Arrays

(Es werden nur Arrays mit bis zu 3 Dimensionen unterstützt)

Mehrdimensionale Arrays sind Arrays, die Referenzen zu weiteren Sub-Arrays enthalten. Zum Beispiel ist ein zweidimensionales Array ein "Array auf Eindimensionale Arrays".

Beispiele für die Deklaration von zweidimensionalen Arrays:

```
new a[4][3]
new b[3][2] = [ [ 1, 2 ], [ 3, 4 ], [ 5, 6 ] ]
new c[3][3] = [ [ 1 ], [ 2, ...], [ 3, 4, ... ] ]
new d[2]{10} = [ "agreement", "dispute" ]
new e[2][] = [ ''OK'', ''Cancel'' ]
new f[][] = [ ''OK'', ''Cancel'' ]
```

Wie die beiden letzten Deklarationen (Variablen "e" und "f") zeigen, hat die letzte Dimension eine nicht spezifizierte Länge. In diesem Fall wird die Länge des Sub-Arrays aus dem dazugehörigen Initialisierer erkannt. Jedes Sub-Array hat eine unterschiedliche Länge. In diesem speziellen Beispiel enthält "e[1][5]" den Buchstaben "l" des Wortes "Cancel", aber "e[0][5]" ist ungültig, da das Sub-Array e[0] nur drei Einträge (die Buchstaben "O", "K", und den Null-Terminator) beinhaltet. Der Unterschied zwischen den Deklarationen der Arrays "e" und "f" ist, dass wir bei "f" den Compiler die Anzahl der höheren Dimension ermitteln lassen. "sizeof f" ist 2 genauso wie "sizeof e" (siehe "Arrays und der "sizeof"-Operator" auf Seite 95).

4.4.4.5 Arrays und der "sizeof"-Operator

Der "sizeof"-Operator gibt die Anzahl der Elemente einer Variablen zurück. Für eine einfache (nicht Array) Variable ist das Ergebnis von "sizeof" immer 1.

Ein Array mit einer Dimension enthält eine Anzahl von Elementen und der "sizeof"-Operator gibt diese Anzahl zurück. Der Codeausschnitt unterhalb würde deshalb "5" ausgeben, da das Array 4 Zeichen und den Null-Terminator enthält.

```
new msg[] = ''Help''
printf(''%d'', sizeof msg);
```

Der "sizeof"-Operator gibt immer die Anzahl der Einträge, auch für ein "packed" Array, zurück. Der Codeausschnitt unterhalb gibt auch "5" aus, da die Variable 5 Einträge enthält, auch wenn diese im Speicher weniger Platz benötigt.

```
new msg{} = "Help"
printf(''%d'', sizeof msg);
```

Bei mehrdimensionalen Arrays kann der "sizeof"-Operator die Anzahl der Elemente jeder Dimension zurückgeben. Für die letzte (niedrigste) Dimension ist ein Element ein einzelner Eintrag, jedoch für die höchste Dimension ist es ein Sub-Array. Beachten Sie, dass im nachfolgenden Codeausschnitt die Syntax "sizeof matrix" die Anzahl der Elemente der höhere Dimension zurückgibt, und dass die Syntax "sizeof matrix[]" die niedrigere Dimension des zweidimensionalen Arrays ausgibt. Der Codeausschnitt gibt 3 (höhere Dimension) und 2 (niedrigere Dimension) aus.

```
new matrix[3][2] = { { 1, 2 }, { 3, 4 }, { 5, 6 } }
printf(''%d %d'', sizeof matrix, sizeof matrix[]);
```

Die Anwendung des "sizeof"-Operators auf mehrdimensionale Arrays ist besonders praktisch, wenn er als Standardwert für Funktionsargumente verwendet wird.

4.4.5 Operatoren und Ausdrücke

4.4.5.1 Zeichenerklärung

Die Anwendung von einigen Operatoren hängt von der jeweiligen Art des Operanden ab. Aus diesem Grund wird in diesem Kapitel folgende Notation angewendet:

e
: *beliebiger Ausdruck (eng. expression)*

v
: *beliebiger Ausdruck, dem ein Wert zugewiesen werden kann (“lvalue” Ausdruck - Variable)*

a
: *ein Array*

f
: *eine Funktion*

s
: *ein Symbol - dies kann eine Variable, eine Konstante oder eine Funktion sein*

4.4.5.2 Ausdrücke

Ein Ausdruck besteht aus ein oder mehreren Operanten mit einem Operator. Der Operand kann eine Variable, eine Konstante oder ein anderer Ausdruck sein. Ein Ausdruck gefolgt von einem Semikolon ist ein Statement.

Beispiele für Ausdrücke:

```
v++ f(a1, a2)
v = (ia1 * ia2) / ia3
```

4.4.5.3 Arithmetik

Operator	Beispiel	Erklärung
+	e1 + e2	Ergebnis der Addition von e1 und e2
-	e1 - e2	Ergebnis der Subtraktion von e1 und e2
	-e	Ergebnis der arithmetischen Negation von e (Zweierkomplement)
*	e1 * e2	Ergebnis der Multiplikation von e1 und e2
/	e1 / e2	Ergebnis der Division e1 durch e2. Das Ergebnis wird zum nächstgelegenen ganzzahligen Wert, der kleiner oder gleich dem Quotienten ist, abgeschnitten. Sowohl positive als auch negative Werte werden abgerundet (Richtung unendlich).
%	e1 % e2	Ergebnis ist der Rest der Division e1 durch e2. Das Vorzeichen ist dasselbe wie bei e2
++	v++	erhöht v um 1. Das Ergebnis des Ausdrucks ist der Wert vor der Erhöhung.
	++v	erhöht v um 1. Das Ergebnis des Ausdrucks ist der Wert nach der Erhöhung.
--	v--	verringert v um 1. Das Ergebnis des Ausdrucks ist der Wert vor der Verringerung.
	--v	verringert v um 1. Das Ergebnis des Ausdrucks ist der Wert nach der Verringerung.

Hinweis: *Das unäre + ist in Pawn nicht definiert. Die Operatoren ++ und -- ändern den Operanden. Der Operand muss ein "lvalue" sein.*

4.4.5.4 Bit-Manipulation

Operator	Beispiel	Erklärung
~	~e	Ergebnis ist das Einerkomplement von e.
››	e1 ›› e2	Ergebnis der arithmetischen Verschiebung nach rechts von e1 durch e2 Bits. Die Verschiebung ist vorzeichenbehaftet: Das Bit ganz links wird auf die freien Bits des Ergebnisses kopiert.
›››	e1 ››› e2	Ergebnis der logischen Verschiebung nach rechts von e1 durch e2 Bits. Die Verschiebung ist vorzeichenlos. Die freien Bits des Ergebnisses werden mit o aufgefüllt.
‹‹	e1 ‹‹ e2	Ergebnis: Verschiebung nach links von e1 durch e2 Bits. Die freien Bits des Ergebnisses werden mit o aufgefüllt. Es gibt keinen Unterschied zwischen einer arithmetischen und einer logischen Verschiebung nach links.
&	e1 & e2	Ergebnis ist das bitweise logische "und" von e1 und e2.
|	e1 | e2	Ergebnis ist das bitweise logische "oder" von e1 und e2.
^	e1 ^ e2	Ergebnis ist das bitweise "exklusiv oder" von e1 und e2.

4.4.5.5 Zuweisung

Das Ergebnis eines Zuweisungsausdrucks ist der Wert des Operanden nach der Zuweisung.

Operator	Beispiel	Erklärung
=	v = e	weist den Wert von e der Variable v zu
	v = a	weist das Array a der Variable v zu. v muss ein Array mit derselben Größe und denselben Dimensionen sein wie a. a kann eine Zeichenkette oder ein Array sein.

Hinweis: Die folgenden Operatoren kombinieren eine Zuweisung mit einer arithmetischen oder bitweisen Operation. Das Ergebnis des Ausdrucks ist der Wert des linken Operanten nach der arithmetischen oder bitweisen Operation.

Operator	Beispiel	Erklärung
+=	v += e	erhöht v um e
-=	v -= e	vermindert v um e
*=	v *= e	multipliziert v mit e
/=	v /= e	dividiert v mit e
%=	v %= e	weist v den Rest der Division von v und e zu
››=	v ››= e	verschiebt v arithmetisch um e Bits nach rechts
›››=	v ›››= e	verschiebt v logisch um e Bits nach rechts
‹‹=	v ‹‹= e	verschiebt v um e Bits nach links
&=	v &= e	führt ein bitweises "und" von v und e aus und weist das Ergebnis v zu
|=	v |= e	führt ein bitweises "oder" von v und e aus und weist das Ergebnis v zu
^=	v ^= e	führt ein bitweises "exklusiv oder" von v und e aus und weist das Ergebnis v zu

4.4.5.6 Vergleichsoperatoren

Ein logisches "false" wird durch einen Integer-Wert von 0 repräsentiert; ein logisches "true" durch einen Wert, der nicht 0 ist. Ergebnisse eines Vergleichs-Ausdrucks sind entweder 0 oder 1 und ihr "tag" wird auf "bool" gesetzt.

Operator	Beispiel	Erklärung
==	e1 == e2	Das Ergebnis ist "true", wenn e1 und e2 gleich sind.
!=	e1 != e2	Das Ergebnis ist "true", wenn e1 nicht gleich e2 ist.

Hinweis: *Die folgenden Operatoren können verkettet werden, wie im Ausdruck "e1 <= e2 <= e3". Dies bedeutet, dass das Ergebnis "1" ist, wenn jeder einzelne Vergleich zutrifft und "0", wenn zumindest ein Vergleich nicht zutrifft.*

Operator	Beispiel	Erklärung
<	e1 < e2	Das Ergebnis ist ein logisches "true", wenn e1 kleiner ist als e2.
<=	e1 <= e2	Das Ergebnis ist ein logisches "true", wenn e1 kleiner oder gleich e2 ist.
>	e1 > e2	Das Ergebnis ist ein logisches "true", wenn e1 größer ist als e2.
>=	e1 >= e2	Das Ergebnis ist ein logisches "true", wenn e1 größer oder gleich e2 ist.

4.4.5.7 Boolean

Ein logisches "false" wird durch einen Integer-Wert von 0 repräsentiert, ein logisches "true" durch einen Wert, der nicht 0 ist. Ergebnisse eines Vergleichs-Ausdrucks sind entweder 0 oder 1 und ihr "tag" wird auf "bool" gesetzt.

Operator	Beispiel	Erklärung
!	!e	Das Ergebnis ist eine logisches "true", wenn e logisch "false" ist.
||	e1 || e2	Das Ergebnis ist "true", wenn entweder e1 oder e2 (oder beide) logisch "true" sind. Der Ausdruck e2 wird nur ausgewertet, wenn e1 logisch "false" ist.
&&	e1 && e2	Das Ergebnis ist "true", wenn e1 und e2 logisch "true" sind. Der Ausdruck e2 wird nur ausgewertet, wenn e1 logisch "true" ist.

4.4.5.8 Sonstiges

Operator	Beispiel	Erklärung
[ ]	a[e]	Array Index: Das Ergebnis ist der Eintrag an der Position e des Arrays a.
{ }	a{e}	Array Index: Das Ergebnis ist das Zeichen an der Position e des "packed" Arrays a.
( )	f(e1, e2, ... eN)	Das Ergebnis ist der Wert, der von der Funktion f zurückgegeben wird. Die Funktion wird mit den Parametern e1, e2, ... eN aufgerufen. Die Reihenfolge der Auswertung der Parameter ist nicht definiert. (Die Implementation der Script-Engine wertet die Parameter möglicherweise in umgekehrter Reihenfolge aus.)
? :	e1 ? e2 : e3	Das Ergebnis ist entweder e2 oder e3, abhängig vom Wert von e1. Der bedingte Ausdruck ist ein zusammengesetzter Ausdruck mit einem zweiteiligen Operator, "?" und ":". Der Ausdruck e2 wird ausgewertet, wenn e1 logisch "true" ist, e3 wird ausgewertet, wenn e1 logisch "false" ist.
:	tagname: e	"tag" Überschreibung: Der Wert des Ausdrucks ändert sich nicht, jedoch ändert sich der "tag".
defined	defined s	Ergebnis ist "1", wenn das Symbol definiert wurde. Das Symbol kann eine Konstante oder eine globale oder lokale Variable sein. Der "tag" des Ausdrucks ist "bool".
sizeof	sizeof s	Das Ergebnis ist die Anzahl der Elemente der angegeben Variable. Für einfache Variablen und für eindimensionale Arrays ist ein Element ein Eintrag. Für mehrdimensionale Arrays ist das Ergebnis die Anzahl der Elemente (Sub Arrays) in der höchsten Dimension. Fügen Sie [] zum Namen des Arrays hinzu, um eine niedrigere Dimension anzugeben. Wenn die Größe der Variable nicht bekannt ist, dann ist das Ergebnis 0. Wenn dieser Operator in einem "default"-Wert einer Funktion verwendet wird, dann wird der Ausdruck zum Zeitpunkt des Aufrufs der Funktion und nicht zum Zeitpunkt der Definition ausgeführt.
tagof	tagof s	Das Ergebnis ist eine eindeutige Zahl, die den "tag" der Variablen, der Konstanten, des Rückgabewerts einer Funktion oder des Names der "tag"-Bezeichnung repräsentiert. Wenn dieser Operator in einem "default"-Wert einer Funktion verwendet wird, dann wird der Ausdruck zum Zeitpunkt des Aufrufs der Funktion und nicht zum Zeitpunkt der Definition ausgeführt.

4.4.5.9 Priorität der Operatoren

Die nachfolgende Tabelle gruppiert Operatoren mit derselben Priorität, beginnend mit der höchsten Priorität.

Wenn die Auswertung eines Ausdrucks nicht explizit durch Klammern begründet wird, wird sie von den Assoziationsregeln bestimmt. Zum Beispiel: a*b/c ist gleich (a*b)/c auf Grund der links zu rechts Assoziation, und a=b=c ist gleichzusetzen mit a=(b=c).

Operator	Erklärung	Lesefolge
( ) [ ] { }	Funktionsaufruf Array Index (Element) Array Index (Zeichen)	links-nach-rechts
! ~ - ++ -- : defined sizeof tagof	logisches Nicht Einerkomplement Zweierkomplement (unäres Minus) Erhöhung Verringerung "tag"-Überschreibung Symbol Definitions-Status Symbol Größe in "Elementen" eindeutige Zahl des "tags"	rechts-nach-links
* / %	Multiplikation Division Modulo	links-nach-rechts
+ -	Addition Subtraktion	links-nach-rechts
›› ››› ‹‹	arithemtische Verschiebung nach rechts logische Verschiebung nach rechts Verschiebung nach links	links-nach-rechts
&	bitweises "und"	links-nach-rechts
^	bitweises "exklusiv oder"	links-nach-rechts
|	bitweises "oder"	links-nach-rechts
‹ ‹= › ›=	kleiner als kleiner oder gleich als größer als größer oder gleich als	links-nach-rechts
== !=	gleich ungleich	links-nach-rechts
&&	logisches "und"	links-nach-rechts
||	logisches "oder"	links-nach-rechts
? :	bedingte Ausführung	rechts-nach-links
=	Zuweisung *= /= %= += -= ››= ›››= ‹‹= &= ^= |=	rechts-nach-links
,	Komma	links-nach-rechts

4.4.6 Anweisungen

Ein Statement kann aus einer oder mehreren Zeilen bestehen. Eine Zeile kann zwei oder mehrere Statements enthalten.

Statements zur Ablaufsteuerung (if, if-else, for, while, do-while und switch) können geschachtelt werden.

4.4.6.1 Statement-Etikett

Ein Etikett besteht aus einem Identifizierer gefolgt von einem ":". Ein Etikett ist ein "Sprung-Ziel" eines "goto" Statements.

Jede Anweisung kann mit einem Etikett versehen werden. Es muss dem Etikett ein Statement folgen, dies kann auch ein "leeres Statement" sein.

Der Gültigkeitsbereich eines Etiketts ist die Funktion in der es deklariert wurde d.h. ein "goto"-Statement kann nicht aus der aktuellen Funktion in eine andere Funktion springen.

4.4.6.2 Zusammengesetzte Anweisungen

Eine zusammengesetzte Anweisung (auch Block genannt) ist eine Serie von Null oder mehreren Anweisungen, welche durch Klammern ("{" und "}") umgeben ist. Die schließende Klammer ("}") darf nicht mit einem Semikolon abgeschlossen werden. Jede Anweisung kann durch einen Block ersetzt werden. Eine zusammengesetzte Anweisung, die keine Anweisungen enthält, ist ein Spezialfall und wird "leeres Statement" genannt.

4.4.6.3 Ausdrucksanweisung

Jeder Ausdruck wird zu einem Statement, wenn ein Semikolon (";") angehängt wird. Ein Ausdruck wird auch zu einem Statement, wenn dem Ausdruck bis zum Ende der Zeile nur Leerzeichen folgen und der Ausdruck nicht in der nächsten Zeile weitergeführt werden kann.

4.4.6.4 Leeres Statement

Eine leeres Statement führt keine Anweisungen aus und besteht aus einem Block-Statement ohne Anweisungen, d.h. es besteht aus dem Symbol "{ }". Leere Statements werden in Kontrollflussanweisungen ohne Aktionen eingesetzt (z.B. "while (!iskey()) {}") oder, wenn ein Etikett genau vor einer schließenden Klammer eines Block-Statements definiert wird. Ein leeres Statement endet nicht mit einem Semikolon.

4.4.6.5 assert Ausdruck

bricht das Programm mit einem Laufzeitfehler ab, wenn der Ausdruck logisch "false" ergibt

> ***Hinweis:*** *Dieser Ausdruck schützt vor "unmöglich" oder ungültigen Bedingungen. Im folgenden Beispiel ist eine negative Fibonacci-Zahl ungültig. Die assert-Anweisung markiert diesen Fehler als Programmierer-Fehler. assert-Anweisungen sollten nur Programmierer-Fehler kennzeichnen und niemals Benutzereingaben.*

Beispiel:

```
fibonacci(n)
{
  assert n > 0

  new a = 0, b = 1
  for (new i = 2; i < n; i++)
  {
    new c = a + b
    a = b
    b = c
  }
  return a + b
}
```

4.4.6.6 break

beendet und verlässt das kleinste, umschließende "do"-, "for"- oder "while"-Statement an jedem beliebigen Punkt in der Schleife. Das "break"-Statement bewegt den Programmfluss zum nächsten Statement außerhalb der Schleife.

Beispiel:

```
example(n)
{
  new a = 0

  for(new i = 0; i < n ; i++ )
  {
    a += i

    if(i>10)
      break

    a += 1
  }
  return a
}
```

4.4.6.7 continue

beendet die aktuelle Iteration der kleinsten umschließenden "do"-, "for"- oder "while"-Anweisung und bewegt die Programmsteuerung an den Bedingungsteil der Schleife.

Beispiel

```
example(n)
{
  new a = 0

  for(new i = 0; i < n ; i++ )
  {
    a += i

    if(i>10)
      continue

    a += 1
  }
  return a
}
```

4.4.6.8 do Statement while (Ausdruck)

führt ein Statement aus, bevor der Bedingungsteil (die "while"-Bedingung) evaluiert wird. Das Statement wird wiederholt, solange die Bedingung logisch "true" ist. Das Statement wird zumindest einmal ausgeführt.

Beispiel:

```
example(n)
{
  new a = 0

  do
  {
    a++
  }
  while(n >= 0)

  return a
}
```

4.4.6.9 exit Ausdruck

bricht das Programm ab. Der Ausdruck ist optional, aber wenn er vorhanden ist, muss er in der selben Zeile wie das "exit"-Statement beginnen und enden. Die "exit"-Anweisung gibt den Wert des Ausdrucks zurück an die Hauptanwendung oder gibt Null zurück, wenn kein Ausdruck angegeben wird.

4.4.6.10 for (Ausdruck 1 ; Ausdruck 2 ; Ausdruck 3) Statement

Alle drei Ausdrucke sind optional.

Ausdruck 1:

wird nur einmal ausgewertet, vor Eintritt in die Schleife. Dieser Ausdruck kann zum Initialisieren einer Variablen genutzt werden. Dieser Ausdruck hält auch die Variablendeklaration mittels der "new"-Syntax. Eine Variable, die an dieser Stelle deklariert wird, ist nur innerhalb der Schleife gültig. Es ist nicht möglich einen Ausdruck (mit bereits vorhandenen Variablen) und eine Deklaration von neuen Variablen in diesem Feld zu kombinieren. Es müssen entweder alle Variablen in diesem Feld bereits vorhanden sein, oder sie müssen alle in diesem Bereich deklariert werden.

Ausdruck 2:

Dieser Ausdruck wird vor jedem Durchlauf der Schleife ausgeführt und beendet die Schleife, wenn der Ausdruck logisch "false" zurückgibt. Wenn dieser Ausdruck weggelassen wird, wird das Ergebnis des Ausdrucks 2 als logisch "true" angenommen.

Ausdruck 3:

Dieser Ausdruck wird nach jeder Ausführung des Statements ausgeführt. Die Programmsteuerung bewegt sich von Ausdruck 3 zum Ausdruck 2 für die nächste (bedingte) Iteration der Schleife.

Beispiel:

```
example(n)
{
  new a = 0

  for(new i = 0; i < n; i++)
  {
    a++
  }

  return a
}
```

Das Statement "for (; ;)" ist gleich dem Statement "while (true)".

4.4.6.11 goto Etikett

bewegt die Programmsteuerung (unbedingt) zu der Anweisung, die dem angegebenen Etikett folgt. Das Etikett muss innerhalb der gleichen Funktion wie die "goto"-Anweisung sein. Eine "goto"-Anweisung kann nicht aus einer Funktion springen.

4.4.6.12 if (Ausdruck) Statement 1 else Statement 2

führt das Statement 1 aus, wenn das Ergebnis des Ausdrucks logisch "true" ergibt. Die "else"-Klausel des "if"-Statements ist optional. Wenn das Ergebnis des Ausdruck logisch "false" ergibt und eine "else"-Klausel existiert, dann wird das Statement, das mit der "else"-Klausel assoziiert ist, (Statement 2) ausgeführt.

Beispiel:

```
example(n)
{
  if(n < 0)
    return -1
  else if (n == 0)
    return 0
  else
    return 1
}
```

4.4.6.13 return Ausdruck

beendet die aktuelle Funktion und bewegt die Programmsteuerung zum nächsten Statement nach dem Funktionsaufruf. Der Wert des Ausdrucks wird als Funktionsergebnis zurückgeliefert. Der Ausdruck kann ein Array oder eine Zeichenfolge sein. Der Ausdruck ist optional, wenn er jedoch vorhanden ist, muss er an der selben Zeile beginnen, wie das "return"-Statement. Wenn kein Ausdruck angegeben ist, dann wird Null zurückgeliefert.

4.4.6.14 switch (Ausdruck) { case Liste }

überträgt die Ablaufsteuerung an unterschiedliche Statements innerhalb des "switch" in Abhängigkeit vom Wert des "switch"-Ausdrucks. Der Hauptteil der "switch"-Anweisung ist eine zusammengesetzte Anweisung, die eine Reihe von "case"-Klauseln enthält. Jede "case"-Klausel beginnt mit dem Schlüsselwort "case", gefolgt von einer Liste von Konstanten und einem Statement. Die Liste der Konstanten ist eine Serie von Ausdrücken, getrennt durch Kommas, die jeweils zu einem konstanten Wert ausgewertet werden. Diese Liste endet mit einem Doppelpunkt. Um einen Bereich in dieser Liste anzugeben, trennen Sie die untere und obere Grenze des Bereichs mit einem doppelten Punkt (".."). Ein Beispiel für einen Bereich ist: "case 1..9:".

Das "switch"-Statement bewegt die Ablaufsteuerung zu einer "case"-Klausel, wenn ein Wert der Liste dem Wert des "switch"-Ausdrucks entspricht.

Die "default"-Klausel besteht aus dem Schlüsselwort "default" und einem Doppelpunkt. Die "default"-Klausel ist optional, aber wenn sie angegeben wird, muss sie als letzter Eintrag in der "case"-Liste eingetragen sein. Das "switch"-Statement bewegt die Ablaufsteuerung zur "default"-Klausel, wenn keine der "case"-Klauseln mit dem "switch"-Ausdruck übereinstimmt.

Beispiel:

```
example(n)
{
  new a = 0

  switch (n)
  {
    case 0..3:
      a = 0
    case 4,6,8,10:
      a = 1
    case 5,7:
      a = 2
    case 9:
      a = 3
    default:
      a = -1
  }

  return a
}
```

4.4.6.15 while (Ausdruck) Statement

wertet den Ausdruck aus und führt das Statement aus, wenn das Ergebnis des Ausdrucks logisch "true" ergibt. Nachdem die Anweisung ausgeführt wurde, kehrt die Programmsteuerung erneut zu dem Ausdruck zurück. Das Statement wird daher ausgeführt, solange der Ausdruck logisch "true" ist.

Beispiel:

```
example(n)
{
  new a = 0

  while(n >= 0)
  {
    a++
  }

  return a
}
```

4.4.7 Funktionen

Eine Funktionsdeklaration spezifiziert den Namen der Funktion und die formalen Parameter, die in Klammern eingeschlossen sind. Eine Funktion kann auch einen Wert zurückliefern. Eine Funktion

muss global definiert, d.h. außerhalb einer anderen Funktion deklariert werden und ist global verfügbar.

Wenn ein Semikolon der Funktionsdeklaration folgt (anstatt einer Anweisung), dann ist dies eine Vorwärtsdeklaration einer Funktion.

Die "return"-Anweisung setzt den Rückgabewert der Funktion. Zum Beispiel, hat die Funktion "sum" (siehe unten) als Rückgabewert die Summe der beiden Parameter. Der "return"-Ausdruck ist optional.

```
sum(a, b)
{
  return a + b
}
```

Argumente einer Funktion sind (implizit deklarierte) lokale Variablen für diese Funktion. Der Funktionsaufruf bestimmt die Werte der Argumente. Ein weiteres Beispiel für eine vollständige Definition einer Funktion ist "leapyear", die "true" für ein Schaltjahr und "false" für kein Schaltjahr zurückgibt.

```
leapyear(y)
{
  return y % 4 == 0 && y % 100 != 0 || y % 400 == 0
}
```

Die Anweisungen, die in diesem Beispiel verwendet wurden, werden im Kapitel "Operatoren und Ausdrücke" auf Seite 96 behandelt.

Normalerweise beinhalten Funktionen lokale Variablendeklarationen und bestehen aus einer Block-Anweisung.

> ***Hinweis:*** *Im nächsten Beispiel verhindert die "assert"-Anweisung negative Werte für den Exponenten*

```
power(x, y)
{
  /* gibt x hoch y zurück*/
  assert y >= 0

  new r = 1
  for (new i = 0; i < y; i++)
    r *= x

  return r
}
```

Eine Funktion kann mehrere "return"-Anweisungen enthalten, eine wird z.B. benutzt um schnell eine Funktion zu beenden, wenn ungültige Parameter übergeben werden, oder wenn sich herausstellt, dass die Funktion nichts zu tun hat. Wenn eine Funktion ein Array zurückgibt, müssen alle "return" Anweisungen ein Array mit derselben Anzahl von Einträgen zurückgeben.

4.4.7.1 Funktionsargumente ("call-by-value" versus "call-by-reference")

Die "faculty"-Funktion im nächsten Beispiel hat einen Parameter, der in der Schleife benutzt wird um die Fakultät dieser Zahl zu berechnen. Was Aufmerksamkeit verdient ist, dass die Funktion das Argument modifiziert.

```
main()
{
  new v = 5
  new f = faculty(v)
}

faculty(n)
{
  assert n >= 0

  new result = 1
  while (n > 0)
    result *= n--

  return result
}
```

Egal welchen (positiven) Wert die Variable "n" am Beginn der "while"-Schleife hatte, am Ende der Funktion wird "n" Null sein. Am Beispiel der Funktion "faculty" wird der Parameter als Wert ("by value") übergeben, damit ist eine Änderung der Variable "n" nur lokal in der Funktion "faculty" gültig. Mit anderen Worten, die Variable "v" in der Funktion "main()" hat vor und nach dem Aufruf der Funktion denselben Wert.

Argumente können als Wert ("by value") oder als Referenz ("by reference") übergeben werden. Einem Funktionsargument, das als Referenz übergeben werden soll, muss als Präfix "&" dem Namen vorangestellt werden. Als Standard werden der Funktion die Argumente als Wert übergeben.

Beispiel:

```
swap(&a, &b)
{
  new temp = b
  b = a
  a = temp
}
```

Um eine Array einer Funktion zu übergeben, fügen Sie ein Klammernpaar ("[]") dem Namen des Arguments an. Es kann auch zusätzlich die Anzahl der Einträge angegeben werden. Dies verbessert die Fehlererkennung des Parsers des Compilers.

Beispiel:

```
addvector(a[], const b[], size)
{
  for (new i = 0; i < size; i++)
    a[i] += b[i]
}
```

Arrays werden immer als Referenz übergeben.

> ***Hinweis:*** *Das Array "b" im oben gezeigten Beispiel wird in der Funktion nicht verändert. Dieses Funktionsargument wurde als "const" deklariert, um dies explizit zu machen. Zusätzlich zur verbesserten Fehlererkennung, erlaubt es dem Compiler einen effizenteren Code zu generieren.*

Das folgende Codebeispiel ruft die Funktion "addvector" auf und addiert zu jedem Element der Variablen "vect" den Wert 5:

```
new vect[3] = [ 1, 2, 3 ]

addvector(vect, [5, 5, 5], 3)

/* vect[] beinhaltet nun die Werte 6, 7 und 8 */
```

4.4.7.2 Benannte Parameter versus positionsgebundene Parameter

In den vorangegangenen Beispielen war die Reihenfolge der Parameter bei einem Funktionsaufruf wichtig, da jeder Parameter zu dem Funktionsparameter mit derselben Position kopiert wurde. Zum Beispiel bei der "weekday" Funktion, die unterhalb definiert wird, würde der Aufruf "weekday(12, 31, 1999)" lauten, um den Wochentag des letzten Tages des letzten Jahrhunderts zu erhalten.

```
weekday(month, day, year)
{
  /* gibt den Tag der Woche zurück: 0=Samstag, 1=Sonntag, etc. */
  if (month <= 2)
    month += 12, --year

  new j = year % 100
  new e = year / 100
  return (day + (month+1)*26/10 + j + j/4 + e/4 - 2*e) % 7
}
```

Das Datumsformat unterscheidet sich je nach Kultur und Nation, während in den Vereinten Staaten von Amerika das Format Monat/Tag/Jahr verbreitet ist, verwenden europäische Staaten oft das Format Tag/Monat/Jahr und in technischen Publikationen wird Jahr/Monat/Tag (ISO/IEC 8824) verwendet. Mit anderen Worten, keine Reihenfolge der Parameter ist "standardisiert" oder "normal". Aus diesem Grund gibt es eine alternative Möglichkeit, um Parameter an eine Funktion zu übergeben: die "benannten Parameter". Diese wird im nächsten Beispiel gezeigt (die Funktion wurde genau wie im vorherigen Beispiel deklariert).

```
new wkday1 = weekday( .month = 12, .day = 31, .year = 1999)
new wkday2 = weekday( .day = 31, .month = 12, .year = 1999)
new wkday3 = weekday( .year = 1999, .month = 12, .day = 31)
```

Bei "benannten Parametern" wird ein Punkt (".") dem Namen des Arguments vorangestellt. Das Argument der Funktion kann auf einen beliebigen Ausdruck gesetzt werden, der gültig für das Argument ist. Das Gleichheitszeichen ("=") hat im Falle eines benannten Parameters nicht die Bedeutung einer Zuordnung, sondern verknüpft den Ausdruck mit einem Funktionsargument.

Es können positionsgebundene und benannte Parameter vermischt werden, jedoch müssen die positionsgebundenen vor den benannten Parametern angegeben werden.

4.4.7.3 Standardwerte von Funktionsargumenten

Ein Funktionsargument kann einen Standardwert haben. Der Standardwert eines Funktionsarguments muss eine Konstante sein. Um einen Standardwert anzugeben, fügen Sie an den Namen des Parameters ein Gleichheitszeichen ("=") und den Wert an.

Wenn bei einem Funktionsaufruf ein Platzhalter anstelle eines gültigen Funktionsparameters angegeben wird, wird der Standardwert übernommen. Der Platzhalter ist das Unterstrichzeichen ("_"). Der Argumentplatzhalter ist nur für Parameter mit einem Standardwert gültig.

Die rechten Argumentplatzhalter können von der Argumentenliste entfernt werden.

Zum Beispiel, wenn die Funktion "increment" wie folgt definiert ist:

```
increment(&value, incr=1)
{
  value += incr
}
```

sind die folgenden Funktionsaufrufe alle gleich:

```
increment(a)
increment(a, _)
increment(a, 1)
```

Standardwerte für Argumente, die als Referenz übergeben werden, sind hilfreich um diese Parameter optional zu machen. Zum Beispiel, wenn die Funktion "divmod" geschrieben wurde, um sowohl den Quotienten als auch den Rest als Parameter zu übergeben.

```
divmod(a, b, &quotient=0, &remainder=0)
{
  quotient = a / b
  remainder = a % b
}
```

Mit der vorangegangenen Definition der Funktion "divmod" sind die folgenden Funktionsaufrufe alle gültig:

```
new p, q

divmod(10, 3, p, q)
divmod(10, 3, p, _)
divmod(10, 3, _, q)
divmod(10, 3, p)
divmod 10, 3, p, q
```

Das nächste Beispiel addiert die Werte von einem Array zu einem anderen. Wenn nur ein Parameter angegeben wird, dann werden die Werte des Arrays um 1 erhöht:

```
addvector(a[], const b[] = {1, 1, 1}, size = 3)
{
  for (new i = 0; i < size; i++)
    a[i] += b[i]
}
```

4.5 Beispiele

t.b.d.

4.6 Unterschiede zu C

- Pawn fehlt der Eingabe-Mechanismus von C. Pawn ist eine "integer-only" Variante von C. Es gibt keine Strukturen oder Unions. Floating Point-Unterstützung muss mit benutzerdefinierten Operatoren und der Hilfe von nativen Funktionen implementiert werden.
- Die Syntax für Gleitkommawerte ist strenger als die in C. Werte wie ".5" und "6." sind in C akzeptabel, aber im Pawn muss man "0.5" und "6.0" schreiben. In C ist der Dezimalpunkt optional, wenn ein Exponent enthalten ist, so kann man in C "2E8" schreiben; Pawn akzeptiert den Großbuchstaben "E" nicht. Verwenden Sie den Kleinbuchstaben "e". Es erfordert das Komma: z.B. "2.0e8" (siehe "Numerische Konstanten" auf Seite 91).
- Pawn unterstützt keine "Zeiger". Für die Übergabe von Funktionsparametern als Referenz bietet Pawn ein "Referenz"-Argument (siehe "Funktionsargumente ("call-by-value" versus "call-by-reference")" auf Seite 109). Das "Platzhalter"-Argument ersetzt einige Verwendungen des NULL-Zeigers (siehe "Standardwerte von Funktionsargumenten" auf Seite 111).
- Zahlen können mit Hexadezimal-, Dezimal-, oder Binärbasis angegeben werden. Die Oktale Basis wird nicht unterstützt (siehe "Numerische Konstanten" auf Seite 91). Hexadezimale Zahlen müssen mit "0x" ("x" in Kleinbuchstaben)beginnen. Das Präfix “0X” ist ungültig.

- "Cases" in einem "switch"-Statement sind nicht "durchfallend". Es muss dem "case"-Label zumindest eine Anweisung folgen. Um mehrere Anweisungen auszuführen, müssen Sie ein zusammengesetztes Statement (mit {}) erstellen (siehe "switch (Ausdruck) { case Liste }" auf Seite 106). In C/C++ ist die "switch"-Anweisung ein "bedingtes goto". In Pawn ist die "switch"-Anweisung ein strukturiertes "if".
- Eine "break"-Anweisung beendet nur Schleifen. In C/C++ beendet die "break"-Anweisung auch ein "case" in einer "switch"-Anweisung.
- Pawn unterstützt "array Zuweisungen", mit der Limitation, dass beide Arrays die gleiche Länge haben müssen. Zum Beispiel, wenn "a" und "b" Arrays mit 6 Zeilen sind, dann ist der Ausdruck "a=b" gültig. Neben Zeichenketten, unterstützt Pawn auch literale Arrays und somit Ausdrücke wie "a = {0,1,2,3,4,5}", wobei "a" eine Array Variable mit 6 Elementen ist.
- "defined" ist ein Operator und keine Präprozessor-Direktive. Der "defined" Operator in Pawn arbeitet mit Konstanten (deklariert mit "const"), globalen Variablen, lokalen Variablen und Funktionen.
- Der "sizeof"-Operator gibt die Größe von Variablen in "Elementen" zurück und nicht in "Bytes". Ein Element ist ein Eintrag oder ein Sub-Array. Weitere Details finden Sie im Kapitel "Sonstiges" auf Seite 100.
- Eine leere Anweisung ist ein leerer Block (mit {}), nicht ein Semikolon (siehe "Zusammengesetzte Anweisungen" auf Seite 102). Diese Änderung verhindert häufige Fehler.
- Eine Division erfolgt in der Weise, dass der Rest der Division das gleiche Vorzeichen hat (oder hätte) wie der Nenner. Bei der Division (Operator "/") erfolgt die Rundung immer zum kleineren ganzzahligen Wert (wobei -2 kleiner ist als -1). D.h. 5/2=2 (2,5 wird zu 2 abgerundet), -5/2=-3 (-2,5 wird zu -3 abgerundet). Der "%"-Operator ergibt immer ein positives Ergebnis unabhängig vom Vorzeichen des Zählers (siehe "Operatoren und Ausdrücke" auf Seite 96).
- Es gibt keinen unären Operator "+", da dieser sowieso ein "no-operation"-Operator ist ("a = +1" ist nicht gültig; korrekt: "a = 1").
- Drei der bitweisen Operatoren haben andere Prioritäten als in C. Die Prioritätsstufe des "&", "^" und "|" Operators ist höher als die relationalen Operatoren. Dennis Ritchie erklärte, dass diese Operatoren in C ihre niedrigen Prioritätsstufen bekamen, weil frühe C-Compiler noch nicht über die logischen Operatoren "&&" und "||" verfügten, so dass stattdessen bitweise "&" und "|" verwendet wurden.
- Das Schlüsselwort "const" in Pawn implementiert die "enum" Funktionalität von C.

- In den meisten Fällen sind Vorwärts-Deklarationen von Funktionen (d.h., Prototypen) nicht notwendig. Pawn ist ein 2-Pass-Compiler. Er erkennt alle Funktionen beim ersten Durchlauf und verwendet diese beim zweiten Durchlauf. Benutzerdefinierte Operatoren müssen jedoch vor der Benutzung deklariert werden. Falls vorhanden, müssen Vorwärts-Deklarationen genau mit der Definition der Funktion übereinstimmen. Die Parameternamen in den Prototypen und den Definitionen der Funktionen müssen ident sein. Pawn kümmert sich um Parameter-Namen im Prototyp auf Grund der "benannte Parameter"-Funktion. Pawn verwendet Prototypen, um vorwärts deklariere Funktionen aufzurufen. Um diese dabei mit benannten Parametern zu verwenden, muss der Compiler bereits die Namen der Parameter (und ihre Position in der Parameterliste) kennen. Aus diesem Grund müssen die Parameternamen in den Prototypen mit jenen in den Definitionen übereinstimmen.

Kapitel 5 Connector

Das Grundprinzip des C-Control IoT-Starter Kit 10 ist eine sogenannte Storage-2-Storage-Datenübertragung. Für diese Art der Datenübermittlung müssen weder das C-Control IoT-Starter Kit 10 noch der Server über den logischen Inhalt der Datenblöcke Bescheid wissen. Es geht also nur darum, einen Block Daten von A nach B zu transportieren.

Die von einem C-Control IoT-Starter Kit 10 an den Server übermittelten Daten sind somit frei gestaltbar. Es stehen 1024 Byte pro Datensatz zur Verfügung, die beliebig genutzt werden können. Zudem stehen auch noch 10 voneinander unabhängige Speicherblöcke für Konfigurationsdaten mit je 4000 Byte zur Verfügung, die ebenfalls beliebig genutzt werden können.

Um die von einem C-Control IoT-Starter Kit 10 erhaltenen Daten und Konfigurationen auch in Verbindung mit der myDatanet Oberfläche nutzen zu können (Auswertungen, Visualisierungen, Grafiken, etc.), muss eine Beschreibung des Datenblock- bzw. Konfigurationsblock-Inhalts am Server erfolgen. Der Connector beinhaltet sowohl das Werkzeug für die Beschreibung der Daten, als auch die korrekte Bereitstellung der Daten zur Verwendung mit der Oberfläche.

5.1 Voraussetzungen

> ***Wichtiger Hinweis:*** *Bitte lesen Sie dieses Kapitel sorgfältig durch bevor Sie den Connector verwenden! Es ist wichtig, diese Vorgaben einzuhalten, da die korrekte Funktion des Connector ansonsten nicht gewährleistet werden kann.*

Um den Connector für rapidM2M verwenden zu können, müssen einige Voraussetzungen erfüllt werden, die die grundsätzliche Dynamik von rapidM2M zwar einschränken, jedoch nötig sind, um die Daten auch in myDatanet verfügbar machen zu können.

- Verwendung einer Daten-Präambel (gilt nur für Daten)

  zu Beginn des Datensatzes, um die verschiedenen Nutzdatenblöcke (z.B. Messdaten, Alarme, Log-Einträge) voneinander zu separieren. Es wird empfohlen, die ersten ein oder zwei Bytes hierfür zu verwenden. Die Daten-Präambel wird in den ‹split›-tags der Datenstruktur-Beschreibung verwendet.

  Beispiel für die Verwendung eines Bytes:

  0x01 ... Messdaten
  0x02 ... Alarme
  0x03 Log-Einträge

- Vollständigkeit des Nutzdatenblocks (gilt nur für Daten)

  Nutzdatenblöcke sind immer mit allen Datenfeldern zu befüllen, auch wenn sich nur ein oder zwei Werte seit der letzten Übertragung verändert haben. Die Daten werden vom Connector mit Byte-Offset und Länge aus dem Nutzdatenblock extrahiert, daher müssen sich die Felder immer an der gleichen Position befinden.

- Homogenität des Nutzdatenblocks (gilt nur für Daten)

  Ein Nutzdatenblock der mittels Split-tag (siehe "Split-tag" auf Seite 116) in einen der strukturierten Messdatenkanäle ("histdata0" - "histdata9") kopiert wird, muss immer gleich aufgebaut sein (d.h. die selben Datenfelder enthalten). Werden unterschiedliche Nutzdatenblockstrukturen verwendet, muss jeder Variante mittels Split-tag ein eigener strukturierter Messdatenkanal zugeordnet werden.

- Verwendung von Big Endian (gilt sowohl für Daten als auch für Konfigurationen)

  Im folgenden Beispiel wird die Ganzzahl 439.041.101 als dword (u32) ab Speicheradresse 10000 gespeichert.

	Big Endian			Little Endian		
Adressen	Hex	Dez	Binär	Hex	Dez	Binär
10000	1A	26	00011010	4D	77	01001101
10001	2B	43	00101011	3C	60	00111100
10002	3C	60	00111100	2B	43	00101011
10003	4D	77	01001101	1A	26	00011010

5.2 Split-tag

Ein Split-tag wird verwendet, um die vom Gerät erhaltenen Daten serverseitig richtig zuzuordnen.

Ein C-Control IoT-Starter Kit 10 übermittelt seine Daten im sogenannten "rm2mraw"-Bereich. Dieser Bereich ist für die reine Storage-2-Storage-Verwendung des Servers vorgesehen. Sollen alle oder bestimmte Daten für Auswertungen, Ansichten, Grafiken, etc. in myDatanet verfügbar gemacht werden, muss ein Split-tag konfiguriert werden.

> ***Wichtiger Hinweis:*** *Die Split-tags werden nicht auf bereits im System vorhandene Daten angewandt. Beim Empfang jedes rapidM2M -Rohdatensatzes ("rm2mraw") wird das erste Byte ausgewertet. Bei Werten kleiner/gleich 9 wird der Datensatz, vorausgesetzt es ist kein anders lautender Split-tag vorhanden, automatisch in den jeweiligen strukturierten Messdatenkanal kopiert. Enthält das erste Byte den Wert 0, wird der Datensatz in Messdatenkanal 0 ("histdata0") kopiert. Enthält das erste Byte den Wert 9, wird der Datensatz in Messdatenkanal 9 ("histdata9") kopiert. Eine nachträgliche Verarbeitung ist nicht vorgesehen.*
>
> ***Wichtiger Hinweis:*** *Wollen Sie nur den Rohdatensatzes ("rm2mraw") verwenden, sollten Sie*

dafür Sorge tragen, dass das erste Byte des Rohdatensatzes immer einen Wert größer 9 enthält.

Beispiel für einen Split-tag:

Code	Erklärung
`<split>`	Beginn des Split-tags
`source =rm2mraw`	Quelle der Daten, die kopiert werden sollen
`target =histdata0`	Ziel, in das die Daten kopiert werden sollen
`key =*`	Schlüssel für die Auftrennung der Daten
`</split>`	Ende des Split-tags

source

Tabellenname der Quelle
Bei rapidM2M ist die Quelle immer `rm2mraw`.

target

Tabellenname des Ziels
Die strukturierten Messdatenkanäle sind bei rapidM2M mit "histdata0" - "histdata9" bezeichnet.

key

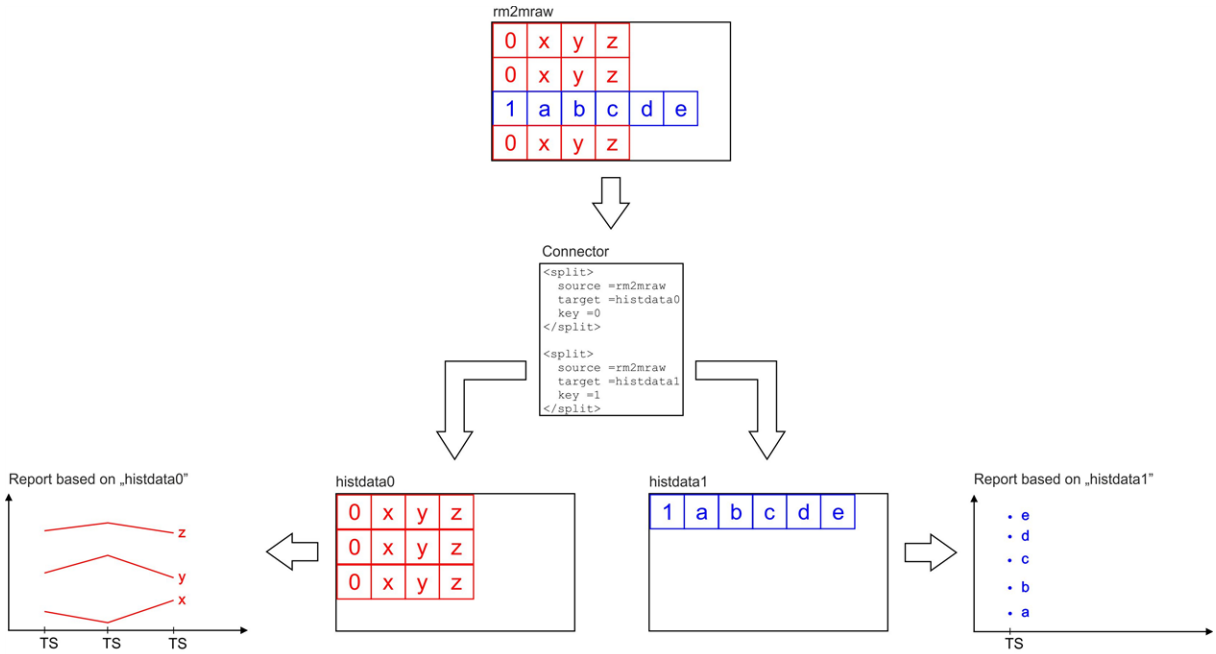
Dieses Feld bietet die Möglichkeit zu bestimmen, ob der aktuelle Datensatz kopiert werden soll:

- *Bei Angabe eines Sterns (*) werden alle Datensätze von source nach target kopiert.*
- *Bei Angabe eines Wertes im HEX-Format wird überprüft, ob der Beginn des empfangenen Datensatzes (auch mehr als das erste Byte bei entsprechender Angabe) diesem Wert entspricht. Bei Übereinstimmung wird dieser Datensatz kopiert.*

Es können mehrere Split-tags angegeben werden. Der Datensatz wird beim ersten passenden key-Eintrag kopiert. Die darauf folgenden Split-tags werden nicht mehr berücksichtigt.

Hinweis: *Ergänzende Erklärung zur Funktionsweise der Split-tags*

Im folgenden Beispiel existieren zwei Nutzdatenblockstrukturen, die mittels Connector vom unstrukturierten Bereich ("rm2mraw") in unterschiedliche strukturierte Messdatenkanäle ("histdata0" und "histdata1") kopiert werden sollen, um sie in Verbindung mit der myDatanet Oberfläche nutzen zu können (Auswertungen, Visualisierungen, Grafiken, etc.). Das erste Byte zu Beginn des Datenbereich eines Nutzdatenblocks wird dazu verwendet, den Typ der Nutzdatenblockstruktur anzugeben. Der Wert 0 gibt an, dass der Nutzdatenblock in den strukturierten Messdatenkanal "histdata0" kopiert werden soll. Ist das erste Byte 1, soll der Nutzdatenblock in den strukturierten Messdatenkanal "histdata1" kopiert werden. Jeder Nutzdatenblock verfügt auch über Header-Informationen (Zeitstempel, CRC,...), die in der folgenden Grafik zugunsten der Übersichtlichkeit weggelassen wurden.



Graphische Darstellung der Funktionsweise der Split-tags

5.3 Datenstruktur

Dieser Abschnitt beschreibt wie strukturierte Messdatenkanäle ("histdata0" - "histdata9") sowie Konfigurationsspeicherblöcke ("config0" - "config9") für die Verwendung am C-Control-Server in einzelne Datenfelder aufgeteilt werden.

> ***Wichtiger Hinweis:*** *Soll ein strukturierter Messdatenkanal oder Konfigurationsblock am C-Control-Server oder über die REST-API verfügbar sein, müssen alle Datenfelder mittels Connector definiert werden.*

Ein erweitertes Beispiel, in dem die meisten der verfügbaren Attribute verwendet werden, finden Sie im Kapitel "Beispiel" auf Seite 123.

Beispiel für die Aufteilung eines strukturierten Messdatenkanals ("histdata0") in einzelne Datenfelder

Code	Erklärung

`<table>`
`  name  =histdata0`
`  <field>`
`  name  =ch0`
`  title  =Verzögerung`
`  units  =sec`
`  type  =u16`
`  byteofs =0`
`  vofs  =10`
`  min  =10`
`  max  =2000`
`  chmode  =3`
`  </field>`
`</table>`
	Beginn des Table-tags
	Name der zu beschreibenden Datentabelle
	Beginn einer Feld-Definition (cho) für ein Feld des Typs u16, Beschreibung der Attribute siehe unten
	
	Ende einer Feld-Definition (cho)

Beispiel für die Aufteilung eines Konfigurationsspeicherblocks ("configo") in einzelne Datenfelder

Code	Erklärung

`<table>`
`  name  =config0`
`  <field>`
`  name  =field0`
`  title  =Aufzeichnungsitv`
`  units  =s`
`  type  =u32`
`  byteofs=0`
`  </field>`
`</table>`
	Beginn des Table-tags
	Name der zu beschreibenden Konfigurationstabelle
	Beginn einer Feld-Definition (fieldo) für ein Feld des Typs u32, Beschreibung der Attribute siehe unten
	
	Ende einer Feld-Definition (fieldo)

5.3.1 Attribute der Feld-Definition

name

Alphanumerisch.

- *Bei strukturierten Messdatenkanälen ("histdatao" - "histdata9"):*

  Interner Name des Kanals. Bei rapidM2M sind 1000 Kanäle verfügbar. Es können also die Bezeichnungen cho bis ch999 verwendet werden.

- *Bei Konfigurationsspeicherblöcken ("configo" - "config9"):*

  Interner Name des Konfigurationsparameters. Bei rapidM2M sind 1000 Konfigurationsparameter verfügbar. Es können also die Bezeichnungen fieldo bis field999 verwendet werden.

alias

Alphanumerisch. Alias für den internen Namen des Kanals der von der REST-API verwendet wird.

title

Alphanumerisch. Titel des Feldes. Diese Bezeichnung findet sich danach in allen Auswertungen, Grafiken, etc. für diesen Kanal.
Die Länge von title sollte 16 Zeichen nach Möglichkeit nicht überschreiten, da sonst Darstellungsprobleme an der Oberfläche auftreten können.

units

Alphanumerisch. Einheiten des Werts.
Die Länge von units sollte 8 Zeichen nach Möglichkeit nicht überschreiten, da sonst Darstellungsprobleme an der Oberfläche auftreten können.

type

Datentyp des Feldes. Der Datentyp bestimmt gleichzeitig die Anzahl der aus dem Datenstream herauszuholenden Bytes. Die einzige Ausnahme ist der Typ string (bzw. char), dessen Länge durch das Attribut max bestimmt wird.
Verfügbare Datentypen: s8, s16, s32, s64, u8, u16, u32, f32, f64, string
Alternative Bezeichnungen (standardisierte Werte in Klammer): bint(s8), wint(s16), dint(s32), qint(s64), byte(u8), word(u16), dword(u32), float32(f32), float(f32), float64(f64), char(string)

param1

Alphanumerisch. Momentan einzig erlaubter Wert: nocodepage
Dieses Attribut muss für Felder gesetzt werden, die einen ANSI-kodierten String enthalten.

byteofs

Numerisch, ganzzahlig positiv. Offset des ersten für das Feld zu verwendende Byte, 0-basierend.

bitmask

Hexadezimal, ohne führendes "0x". Bitmaske, um die tatsächlich zu verwendenden Bits aus dem Datenblock heraus zu maskieren, hexadezimal. Nach der Maskierung wird der extrahierte Wert auf das LSB ausgerichtet (LSB-aligned).
Beispiel: Ein u16-Feld wird aus zwei Bytes mit dem Inhalt 0xF3A7 erzeugt. Als Bitmaske wird 0FF0 angegeben. Die Bits werden extrahiert und ans LSB ausgerichtet. Der resultierende HEX-Wert ist also 0x3A (=58 dezimal).

editmask

Formatanweisung für die Anzeige des Feldinhalts auf der Oberfläche des C-Control-Servers bzw. die Eingabe über die Oberfläche des C-Control-Servers

Formatanweisung	*Erklärung*
0=off;1=on	*Es wird eine ComboBox erstellt in der für jeden Eintrag anstelle des Wertes der Text nach dem "=" angezeigt wird. Die Einträge sind durch ";" zu trennen.*

decpl

Numerisch, ganzzahlig positiv. Anzahl der anzuzeigenden Dezimalstellen.

vscale

Numerisch, Fließkomma. Virtuelle Skalierung des Werts. Der extrahierte Wert wird mit diesem Faktor multipliziert und dann erst weiterverwendet.
*Dieser Wert stellt den Wert k aus der Formel k*x+d dar.*

vofs

Numerisch, Fließkomma. Virtueller Offset des Werts. Der extrahierte Wert wird erst mit vscale multipliziert, danach wird vofs zum Wert addiert.
*Dieser Wert stellt den Wert d aus der Formel k*x+d dar.*

min

Der Minimalwert für die weitere Darstellung am Server (z.B. Grafik).

max

Der Maximalwert für die weitere Darstellung am Server (z.B. Grafik). Beim Datentyp string (bzw. char) wird dieser Wert für die Länge der Zeichenfolge herangezogen. Damit ist die Angabe des max-Wertes bei string (bzw. char) verpflichtend.

chmode

Der Kanalmodus. Die hier gewählte Einstellung nimmt Einfluss auf die weitere Verarbeitung und Darstellung des Kanals in den einzelnen Servermodulen.
Es sind folgende Kanalmodi verfügbar:

Modus	*Name*	*Erklärung*
1	*Digital*	*Der Kanal wird als Digitalkanal verarbeitet. Um Probleme zu vermeiden, sollte der enthaltene Wert 0 oder 1 sein.*
2	*Tageszähler*	*ein Zähler, dessen Wert einmal täglich zurückgesetzt wird. Diese Rücksetzung ist durch das Steuerprogramm vorzunehmen!*
3	*Intervallzähler*	*Ein Zähler, der bei jeder Erstellung eines Messdatensatzes zurückgesetzt wird. Diese Rücksetzung ist durch das Steuerprogramm vorzunehmen!*
6 (Standard)	*Analog*	*ein "einfacher" Messwert, z.B. Temperatur*
12	*Endloszähler*	*ein Zähler, dessen Wert nie zurückgesetzt wird, z.B. Wasserzähler, Stromzähler*

index

wird für die benutzerdefinierte Sortierung der Felder in den Auswahllisten verwendet. Der Standardwert für die 1000 verfügbaren Kanäle ist -1, dadurch wird der Kanal ausgeblendet. Sobald ein Kanal in der Datenstruktur-Beschreibung verwendet wird (Es reicht ein einfacher Field-Tag mit dem Attribut name), wird der Wert Index im Hintergrund automatisch auf den Feld-Index gesetzt (Bsp.: bei ch2 wird Index=2).
Diese Standard-Sortierung kann durch die Angabe des Index-Feldes übersteuert werden.

view

Numerisch, ganzzahlig, positiv. Gibt an, ab welchem Benutzerlevel das Feld auf der Oberfläche des C-Control-Servers sichtbar ist.

edit

Numerisch, ganzzahlig, positiv. Gibt den Benutzerlevel an, der erforderlich ist, um den Feldinhalt über die Oberfläche des C-Control-Servers verändern zu können. Wird dieses Attribut nicht angegeben oder ist der angegebene Wert kleiner als jener für das Attribut "view", ist zum Ändern des Feldinhalts der für das Attribut "view" angegebene Benutzerlevel erforderlich.
Soll ein Ändern des Feldinhalts über die REST-API verhindert werden, muss der Wert dieses Attributs auf 99 gesetzt werden.

5.4 Beispiel

`<table>`	Beginn des Table-Tags
`  name   =histdata0`	Name der zu beschreibenden Datentabelle
`  <field>`	Beginn einer Feld-Definition (cho)
`    name    =ch0`	interner Name des Kanals
`    title   =Verzögerung`	Titel des Kanals
`    units   =sec`	Einheiten
`    type    =u16`	Datentyp, mit u16 werden also 2 Bytes verwendet
`    byteofs =0`	Die ersten beiden Bytes des Datenstreams werden genommen.
`    vofs    =10`	Zum extrahierten Wert wird 10 addiert.
`    min     =10`	Der Minimalwert für die weitere Verwendung ist 10.
`    max     =2000`	Der Maximalwert für die weitere Verwendung ist 2000.
`    chmode  =3`	Es handelt sich um einen Intervallzähler.
`    index   =1`	Sortier-Nummer
`  </field>`	Ende des Field-Tags
`  <field>`	
`    name    =ch1`	interner Name des Kanals
`    title   =Höhe`	Titel des Kanals
`    units   =cm`	Einheiten
`    type    =f32`	Datentyp, Fließkomma 32 bit
`    byteofs =2`	Es werden die Bytes 2 bis 5 genommen.
`    decpl   =2`	2 Nachkommastellen
`    vofs    =0`	Kein Offset
`    vscale  =0.01`	Der extrahierte Wert wir mit 0.01 multipliziert.
`    min     =0`	Der Minimalwert für die weitere Verwendung ist 0.
`    max     =2000`	Der Maximalwert für die weitere Verwendung ist 2000.
`    chmode  =6`	Es handlet sich um einen Analogkanal.
`    index   =0`	Sortier-Nummer
`  </field>`	
`  <field>`	
`    name    =ch2`	interner Name des Kanals
`    title   =Pumpe`	Titel des Kanals
`    type    =u8`	Datentyp, ganzzahlig 8 bit
`    byteofs =6`	Es wird das Byte 6 genommen.
`    bitmask =01`	Es wird das niedrigstwertige Bit (LSB) aus dem Byte heraus maskiert.
`    min     =0`	Der Minimalwert für die weitere Verwendung ist 0.
`    max     =1`	Der Maximalwert für die weitere Verwendung ist 1.
`    chmode  =1`	Es handelt sich um einen Digitalkanal.
`  </field>`	
`  <field>`	
`    name    =ch3`	interner Name des Kanals
`    title   =Info`	Titel des Kanals
`    type    =string`	Datentyp, String, Länge wird im Feld max angegeben.
`    byteofs =10`	Der Text startet beim Byte 10.
`    max     =50`	Der Text ist maximal 50 Zeichen lang.
`    index   =10`	Sortier-Nummer
`  </field>`	
`</table>`	

Bei ch2 fehlt die Angabe von index, daher erhält dieser Kanal automatsich den Index 2. Die Sortierung der Kanäle ist also ch1, cho, ch2, ch3.

5.5 Spezialwerte der Datentypen

Jeder numerische Datentyp unterstützt spezielle Zustände wie z.B. NaN (Not a Number). Wird solch ein Wert am Server erkannt, so wird die in myDatanet übliche Anzeige und Weiterverarbeitung angewandt.

Übersicht der möglichen Werte (unsigned):

Wert/Typ	u8 (byte)	u16 (word)	u32 (dword)
NaN	oxFF	oxFFFF	oxFFFFFFFF
OF	oxFE	oxFFFE	oxFFFFFFFE
UF	oxFD	oxFFFD	oxFFFFFFFD
OL	oxFC	oxFFFC	oxFFFFFFFC
SC	oxFB	oxFFFB	oxFFFFFFFB

Übersicht der möglichen Werte (signed):

Wert/Typ	s8 (bint)	s16(wint)	s32 (dint)	s64 (qint)
NaN	127	32767	2147483647	0x7FFFFFFFFFFFFFFF
OF	126	32766	2147483646	0x7FFFFFFFFFFFFFFE
UF	-126	-32766	-2147483646	0x8000000000000002
OL	-127	-32767	-2147483647	0x8000000000000001
SC	-128	-32768	-2147483648	0x8000000000000000

Übersicht der möglichen Werte (float):

Wert/Typ	f32 (float32)	f64 (float64)
NaN	0x7F8xxxxx	0x7FFxxxxxxxxxxxx
OF	nicht verfügbar	nicht verfügbar
UF	nicht verfügbar	nicht verfügbar
OL	nicht verfügbar	nicht verfügbar
SC	nicht verfügbar	nicht verfügbar

Für f32 und f64 zählen für die Interpretation von NaN nur die ersten 12 Bytes.

Kapitel 6 Technische Daten

Versorgung	Li-Po Akku mit 520mAh
Ladespannung	5VDC über Micro-USB Typ B Buchse
Ausgangsspannung	2,8V , max. 30mA
Eingangsbereich der I/Os	0...2,8V
Abmessungen (BHT)	Modul: 39 x 32 x 6mm Akku: 37 x 31 x 6mm
Gewicht	Modul: 7g Akku: 10g
Antennenanschluss	U.FL
Interface	2 x 10 durchkontaktierte Bohrungen im Raster 2,54mm
Universaleingänge	2 x analog oder digital • 0...2,5V : Auflösung 610µV, max. 2,5V, Bürde 1MΩ • Digital: max. 2,8V, low ‹0,84V, high ›1,96V • Zähler: Impulslänge min. 1ms

Serielle Schnittstellen	2 x 2-Wire UART
	• Baudrate:
	  • UART 1: 600...3.000.000
	  • UART 2: 300...9.600 (Low Energy UART)
	• Stopbits: 1, 2
	• Parität:N, E, O
	• Datenbits: 7, 8
	1 x SPI
	• Modus: SPI-Master
	• Clock-Polarität/Phase einstellbar
	• Clock-Frequenz: 12MHz
	• Interface: 4-wire
	1 x I^2C
	• Modus: Master
	• Clock-Frequenz:
	  • Standard Mode: max. 100kHz
	  • Fast Mode: max. 400kHz

I/O-Leitungen	6 x GPIO
Bluetooth	4.1 kompatibles Low Energy Modul

USB-Schnittstelle	1 x Micro USB 2.0 Slave für die Verbindung mit einem PC und zum Laden des Akkus. Für die Kommunikation mit der C-Control IoT-Starter Kit 10 muss am PC das rapidM2M Toolset installiert sein.
Datenspeicher	448 kB interner Flash-Speicher Die Größe der Datensätze ist variabel (max. 1024 Byte) und wird durch das vom User erstellte Script bestimmt. Der systembedingte Overhead beträgt pro Datensatz 10 Byte.
Konfigurationsspeicher	10 unabhängige Blöcke mit je 4000 Byte
Registrierungsspeicher	4 Blöcke mit je 1kB und vordefinierten Verwendungszwecken zur Ablage gerätespezifischer Daten
Speicher für das PAWN-Binary	256kB (unkomprimierte Größe)
Datenübertragung	2G Quad-Band: • 2G GPRS 900MHz / 1800MHz • 2G GPRS 850MHz / 1900MHz
SIM	Die C-Control IoT-Starter Kit 10 ist mit einem integrierten SIM-Chip versehen.

Kapitel 7 Kontaktinformationen

Elco Industrie Automation GmbH
Benzenmühlstr. 9
71723 Großbottwar
Germany, Europe
Tel. +49 7148 154 99 90
info@elco-automation.de
www.elco-automation.de