Perfection in Automation
www.br-automation.com

APC910 / PPC900
Implementierungsanleitung

Datum: 17. Januar 2014

I Versionsstände

Version	Datum	Kommentar	Bearbeiter
1.00	10.09.2012	Erste Ausgabe.	HOH
1.10	11.01.2013	Erweitert für USV IF Option: • Unterschiede zu APC810 beschrieben (ab Seite 9). • Neu: UPS Service Kommando (ab Seite 39). • Neu: UPS Service Register (ab Seite 46). • BrMtcx.h Listing aktualisiert (ab Seite 59). • Neu: USV Codebeispiele (ab Seite 151). Neu: Hinweis für Übernahme der User Serial ID ergänzt (Seite 19). Neu: Hinweise für unterstützte Module bei modulspezifischen Kommandos (ab Seite 29). Neu: Module Correction Kommando (Seite 34). Module Fan Daten Status Bits korrigiert (Seite 38). Fussnoten für Modulnummern ergänzt (Seite 15).	HOH
1.11	20.03.2013	Lesen der PC Lüfter Statistikwerte ergänzt/korrigiert. UPS Service Kommando korrigiert: Gerätenummer und Adresse waren vertauscht. UPS User Settings Daten und UPS Shutdown Daten korrigiert: die untere Grenze beträgt 10 statt 0 Sekunden. Schreibfehler korrigiert. Lesen der USV Statistik („On Battery“ Stunden und Zyklen) ergänzt: Kommandodaten, Beispiel.	HOH
2.00	09.01.2014	„APC910 IF USV“ geändert auf „USV IF Option“. MTCX Command Status Register: • MTCX Fehlercode 86h ergänzt. • Mode R/W auf RO korrigiert. MTCX Command Param Register: Zielgerät 0 und 2 entfernt und 8 (USV) ergänzt. In den Kommandobeschreibungen ist das Zielgerät nun auch als Nummer angegeben. Flash / EEPROM Service Kommando: Adressbereich genauer definiert (0 bis 3FFFh). Panel Kommandos: Gerätenummerbereich korrigiert (0 bis 15 statt 1 bis 15). Diverse Schreib- und Textfehler korrigiert. Verweise korrigiert und ergänzt. LED Service Kommando: LEDs 64 bis 127 in den Daten ergänzt. Display Service Kommando: Kommandocode korrigiert (48h statt 30h). Watchdog Service Kommando: Zeiteinheit ms ergänzt. Module Info Kommando: Datengröße für Bestellnummer Teil 3 korrigiert (9 statt 16 Bytes) und Module Device ID Daten korrigiert (Hardware Revision entfernt). Module Voltage Kommando: Bit Bereich korrigiert (0 bis 14 statt 0 bis 15) und Hinweise ergänzt für Vin (immer 0) und Wcpu (Leistung des Systems statt der CPU). UPS Status Register: Beschreibung/Fussnoten der Statusflags aktualisiert und korrigiert (z.B. Temperaturalarm bei -30 und +60 statt -42 und +82 °C). UPS Service Kommando: Beschreibung ergänzt. USV Anwendereinstellungen: Beschreibung ergänzt. Codebeispiel für USV Anwendereinstellungen lesen/schreiben: Beschreibung ergänzt. Codebeispiel für USV abschalten (Shutdown): Beschreibung ergänzt. GetFanStatistics Codebeispiel: Parameter FanNumber hinzugefügt. Erweitert für PPC900:	HOH

Version	Datum	Kommentar	Bearbeiter
		• Neu: Displayeinheit (Modul 1) und Bus Lüfter Kit (Modul 12). • RAM Temperaturen können gelesen werden. • MTCX Gerätekennung 0Fh ergänzt. • Kommentare für unterstützte Panelnummern ergänzt. FPGA Headerkennung „P9S3“ für SDL3 Display Link ergänzt. Gerätekennungen für AP830, AP9x3 mit SDL und AP9x3 mit SDL3 ergänzt. Zusatz „resistiv“ für Touch Screen in Funktionsbeschreibungen. BIOS Kennung „S“ für Chipsatz HM76 ergänzt. Module Statistics Kommando: Display Statistics für Displayeinheit und Lüfternummer für Fan Statistics hinzugefügt. GetPanelType Codebeispiel vereinfacht. Neues Codebeispiel für Betriebsstunden und Einschaltzyklen der Displayeinheit lesen.	

Tabelle 1: Versionsstände

II Gestaltung von Sicherheitshinweisen

Die Sicherheitshinweise werden im vorliegenden Dokument wie folgt gestaltet:

Sicherheitshinweis	Beschreibung
Gefahr!	Bei Missachtung der Sicherheitsvorschriften und -hinweise besteht Todesgefahr.
Warnung!	Bei Missachtung der Sicherheitsvorschriften und -hinweise besteht die Gefahr schwerer Verletzungen oder großer Sachschäden.
Vorsicht!	Bei Missachtung der Sicherheitsvorschriften und -hinweise besteht die Gefahr von Verletzungen oder von Sachschäden.
Information:	Wichtige Angaben zur Vermeidung von Fehlfunktionen.

Tabelle 2: Gestaltung von Sicherheitshinweisen

III Inhaltsverzeichnis

1 Einleitung

In diesem Dokument ist beschrieben, wie gerätespezifische Funktionen eines B&R Automation PC 910 (kurz: APC910) und B&R Panel PC 900 (kurz: PPC900) aus einer PC Anwendung bedient werden können.

Auf einem APC910 und PPC900 sind z.B. folgende Funktionen möglich:

- Lesen von Firmwareversionen
- Lesen der Geräteinformationen (Seriennummer, Bestellnummer usw.)
- Lesen von Statistikwerten (Betriebsstunden usw.)
- Lesen von Temperaturen
- Lesen von Lüfterdrehzahlen
- Lesen der Geräteinformationen (Seriennummer, Bestellnummer usw.) von angeschlossenen Automation Panels
- Lesen und Einstellen der Displayhelligkeit der Displayeinheit eines PPC900 und von angeschlossenen Automation Panels
- Lesen der Tastenmatrix der Displayeinheit eines PPC900 und von angeschlossenen Automation Panels
- Schalten der LEDs der Displayeinheit eines PPC900 und von angeschlossenen Automation Panels
- Schalten der Run LED
- Bedienung des Watchdog
- Lesen und Einstellen einer User Serial ID
- Lesen von Statuswerten der USV IF Option
- Lesen und Schreiben von Anwendereinstellungen der USV IF Option
- Shutdown der USV IF Option

Die gerätespezifischen Funktionen werden vorwiegend über den eingebauten Maintenance Controller (MTCX) auf dem APC910 und PPC900 durchgeführt.

Über einen 128 Bytes großen I/O Bereich kann der MTCX über Kommandos bedient werden bzw. stellt gerätespezifische Daten zur Verfügung. Kommandos werden für zeitunkritische Funktionen verwendet. Daten, die oft oder mit kurzen Zugriffszeiten abgefragt werden müssen, werden vom MTCX in Registern bereitgestellt.

Die BIOS Version kann anhand des B&R Vendor Strings aus dem BIOS ROM Speicher gelesen werden.

Hinweis: Die gerätespezifischen Funktionen eines APC910 und PPC900 können unter Windows über das B&R Automation Device Interface (kurz: ADI) angesprochen werden. ADI ist in Form des ADI Treibers in den von B&R erstellten Windows Images (Windows Embedded Standard 2009, Windows Embedded Standard 7) bereits beinhaltet und kann auch nachinstalliert werden. Der Zugriff auf ADI wird durch das ADI Development Kit und ADI .NET SDK unterstützt. ADI Treiber, ADI Development Kit und ADI .NET SDK können Sie von der B&R Homepage www.br-automation.com runterladen.

2 Voraussetzungen

Es wird empfohlen, dass Sie die aktuellsten BIOS und Firmwareversionen auf dem PC installieren. Sie können diese von www.br-automation.com downloaden.

Information:

Für einige Funktionen sind möglicherweise bestimmte BIOS oder Firmwareversionen Voraussetzung. Die notwendigen Versionen sind bei den jeweiligen Register- und Kommandobeschreibungen und Codebeispielen erwähnt.

Für die Kommunikation mit dem MTCX müssen Sie in der PC Anwendung Zugriffe auf I/O Portadressen durchführen können. Für das Lesen des BIOS ROM müssen Sie Zugriffe auf Speicheradressen durchführen können.

Falls das nicht möglich ist, müssen die I/O und Speicherzugriffe abhängig vom verwendeten Betriebssystem eventuell über einen Treiber oder eine andere Systemkomponente durchgeführt werden. Eine Interrupt Bedienung ist nicht notwendig.

Für das Verständnis der in diesem Dokument enthaltenen Codebeispiele sind einfache C Kenntnisse notwendig.

3 Übersicht

Im Kapitel „Kompatibilität mit bisherigen Gerätefamilien“ (Seite 9) finden Sie Hinweise für die Portierung bestehender Funktionen von anderen Gerätefamilien (z.B. APC810).

Im Kapitel „MTCX Schnittstelle“ (ab Seite 11) sind die Grundlagen für die gerätespezifischen Funktionen beschrieben.

Sie können diese auch überspringen und sofort die Funktionen aus dem Kapitel „Codebeispiele“ (ab Seite 58) verwenden. Bitte beachten Sie aber die Hinweise im Kapitel „Codierhinweise“ (ab Seite 55).

4 Kompatibilität mit bisherigen Gerätefamilien

APC910 und PPC900 besitzen im Unterschied zu bisherigen Gerätefamilien wie APC620, PPC700, APC810, PPC800, APC820, PP300/400 und PPC300 eine höhere Modularität.
Es können nicht nur die Funktionen und Informationen der Systemeinheit und von optional angeschlossenen Automation Panels angesprochen werden, sondern es werden auf dem Gerät selbst folgende zusätzliche Module unterstützt:

- Displayeinheit (nur PPC900)
- Buseinheit (optional auf PPC900)
- Display Link (= Monitor / Panel Option, optional, nur APC910)
- Speichermodul 1 (optional)
- Speichermodul 2 (optional)
- Lüfter Kit (optional)
- IF Option 1 (optional)
- IF Option 2 (optional)
- Slide-In 1 (optional)
- Slide-In 2 (optional, nur APC910)
- CPU Board
- Bus Lüfter Kit (optional, nur PPC900)

Um diese Module ansprechen zu können, wurden neue modulspezifische MTCX Kommandos eingeführt.

Sie müssen daher folgendes beachten, wenn Sie Funktionen verwenden wollen, die Sie bereits auf einer anderen Gerätefamilie (z.B. APC810) implementiert haben:

- Mit dem **MTCX_CMD_DEVICE_INFO** Kommando werden die Geräteinformationen (Bestellnummer, Seriennummer etc.) des CPU Board gelesen. Die anderen Geräten wie z.B. einem APC810 entsprechenden Geräteinformationen müssen mit dem neuen **MTCX_CMD_MODULE_INFO** Kommando (siehe Seite 29) von der Systemeinheit gelesen werden.
- Statistikwerte (Betriebsstunden etc.) müssen mit dem neuen **MTCX_CMD_MODULE_STATISTICS** Kommando (siehe Seite 32) Kommando von den einzelnen Modulen gelesen werden.
- Temperaturen können nicht mehr direkt aus MTCX Registern, sondern nur mehr mit dem neuen **MTCX_CMD_MODULE_TEMPERATURE** Kommando (siehe Seite 31) gelesen werden.
- Lüfterdrehzahlen können ebenfalls nicht mehr direkt aus MTCX Registern, sondern nur mehr mit dem neuen **MTCX_CMD_MODULE_FAN** Kommando (siehe Seite 38) gelesen werden.
- Mit dem **UPS Service** Kommando (siehe Seite 39) kann nur ein Shutdown der USV durchgeführt werden – ein Neustart der USV ist nicht möglich. Die Verzögerungszeit für das Abschalten der USV hat als gültigen Wertebereich 10 bis 1200 statt 2 bis 65535 Sekunden.
- Die USV besitzt keine eigene Firmware – die USV Funktion wird vom MTCX mit durchgeführt - es kann daher keine Firmware Version gelesen werden.
- Im **UPS Status** Register (siehe Seite 47) sind einige USV Statusflags nicht relevant.
- Das **UPS Values** Register (siehe Seite 48) beinhaltet nur Batteriespannung und –strom. Batteriekapazität und –laufzeit werden von der USV IF Option nicht unterstützt und können nicht gelesen werden.
- Das **UPS Specials** Register ist nicht vorhanden.

Zusätzlich sind einige für APC910 und PPC900 spezifische Besonderheiten zu beachten:

- Gerätespezifische Spannungswerte können mit dem neuen **MTCX_CMD_MODULE_VOLTAGE** Kommando (siehe Seite 36) gelesen werden.
- Die Display Link FPGA Version kann mit dem **MTCX_CMD_MODULE_HEADER** Kommando (siehe Seite 37) gelesen werden.
- Die MTCX Firmware wird nicht mehr in FPGA und PX32 Anteil unterschieden. Es kann daher keine FPGA Version mehr gelesen werden. Die „gesamte“ MTCX Version kann wie die bisher gelieferte PX32 Version gelesen werden (siehe Seite 86).
- Nach dem Schreiben einer neuen User Serial ID mit dem **Device Info** Kommando (siehe Seite 19) ist kein Neustart erforderlich, damit diese vom MTCX übernommen wird und wieder gelesen werden kann.

Gleich behandelt wie bisher werden u.a. alle Tasten- und LED Funktionen sowie alle Automation Panel spezifischen Funktionen und das Watchdog Handling.

Wenn Sie bereits Funktionen für PP500, APC510 oder APC511 verwenden, beachten Sie bitte folgende Unterschiede:

- Auf dem APC910 und PPC900 werden andere Modulnummern verwendet und die Automation Panels anders adressiert: Panelnummer beginnend mit 0 statt 1.
- Der APC910 besitzt kein eingebautes Display.
- Die Displayeinheit eines PPC900 wird mit Panelnummer 15 statt 0 wie beim PP500 adressiert.
- Auf dem APC910 und PPC900 werden keine Drehgeber und Knotennummer unterstützt.
- Statt einem I/O Board FPGA kann es ein Display Link FPGA geben.

5 MTCX Schnittstelle

5.1 Registerübersicht

Die MTCX Schnittstelle besteht aus mehreren Registern, die ab I/O Portadresse 4100h liegen und insgesamt 128 Bytes umfassen. Die einzelnen Register sind 32 Bit breit. Die Registerdaten werden im INTEL Format abgelegt.

Die Register teilen sich in folgende Hauptbereiche ein:

4100h+ Offset	Registerbereich	Beschreibung
0000h-000Fh	MTCX Configuration Register	Konfiguration der MTCX Schnittstelle und des PC
0010h-0027h	MTCX Command Service Register	Schnittstelle für MTCX Kommandos
0028h-002Fh	Hardware Info Register	gerätespezifische Informationen
0030h-004Fh	Baseboard Service Register	Einstellungen und Daten des Baseboard
0050h-0057h	UPS Service Register	Statuswerte und Betriebsdaten der USV
005Ch-005Fh	Panel Switch Register	Auswahl des Panels für den Panel Service Bereich 60h bis 7Fh
0060h-007Fh	Panel Service Register	Einstellungen des 1. bis 16. Panels

Tabelle 3: MTCX Registerübersicht

Die einzelnen Registerbereiche sind in den nächsten Kapiteln beschrieben.

5.2 MTCX Configuration Register

Die MTCX Configuration Register beinhalten Einstellungen der MTCX Schnittstelle und des PC und liegen ab I/O Port 4100h.

4100h+ Offset	Register	Beschreibung
0000	Version	Version der MTCX Firmware
0004	Config COM	Konfiguration der COM Ports
0008	Config Specials	Spezielle gerätespezifische Einstellungen
000C	Config 2nd	Weitere gerätespezifische Einstellungen

Tabelle 4: MTCX Configuration Register Übersicht

5.2.1 Version Register

Dieses Register beinhaltet die Version der MTCX Firmware und liegt auf I/O Port 4100h.

Version Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	B				A							

ID	Name / Funktion	Default	Mode	Bemerkung
A	Minor Version		RO	Niederwertige Versionsnummer
B	Major Version		RO	Höherwertige Versionsnummer

Tabelle 5: MTCX Version Register

5.2.2 Config COM Register

Dieses Register beinhaltet die Einstellungen der gerätespezifischen Schnittstellen und liegt auf I/O Port 4104h.

Information:

Dieses Register ist für B&R interne Verwendung reserviert.

Config COM Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	E				D				C				F				x	B			A							

ID	Name / Funktion	Default	Mode	Bemerkung
A	I/O Addr		R/W	Definiert die I/O Adresse des Zielgerätes: Wird beim Lesen 03FFh gelesen, so ist das adressierte Gerät nicht vorhanden.
B	Device Activation		R/W	Schaltet das Gerät ein bzw. aus: 0 = Aktiv; 1 = Ausgeschalten
C	Device Index		R/W	Geräte Index auf dem Modul.
D	Device Type		R/W	Geräte Type: 0 = KBC 1 = COM 2 = LPC
E	Module Index		R/W	Modul Index
F	IRQ Number		R/W	Interruptnummer

Tabelle 6: MTCX Config COM Register

5.2.3 Config Specials Register

Mit diesem Register können spezielle gerätespezifische Einstellungen gelesen werden. Das Register liegt auf I/O Port 4108h.

Config Specials Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	B	x	x	x	x	A							

ID	Name / Funktion	Default	Mode	Bemerkung
A	Reserve		RO	
B	Panel Count	0	RO	Anzahl der maximal unterstützten Panels [1]: 0 = 16 Panels 1 = 1 Panel

Tabelle 7: MTCX Config Specials Register

5.2.4 Config 2nd Register

Dieses Register beinhaltet weitere gerätespezifische Einstellungen und liegt auf I/O Port 410Ch.

Config 2nd Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	C								x	x	x	B	A			

ID	Name / Funktion	Default	Mode	Bemerkung
A	Reserve	Fh	RO	
B	Display Found	0	RO	„Display vorhanden“ Kennung: 0 = kein Display, 1 = Display gefunden
C	User Input	00h	R/W	Dieses Byte kann direkt vom Anwender beschrieben werden: Bit 0: Run LED Zustand, 0 = aus, 1 = ein

Tabelle 8: MTCX Config 2nd Register

[1] Auf APC910 und PPC900 wird immer 0 = 16 Panels geliefert.

5.3 MTCX Command Service Register (Kommandoschnittstelle)

Die MTCX Command Service Register bilden die Kommandoschnittstelle des MTCX. Diese steht für allgemeine, zeitunkritische Kommandos zur Verfügung und besteht aus mehreren Registern, die ab I/O Port 4110h liegen.

4110h+ Offset	Register	Beschreibung
0000h	MTCX Command Status	Kommandozustand
0004h	MTCX Command Param	Kommandoparameter
0008h	MTCX Command Data 0	Kommandodaten: Byte 0 bis 3
000Ch	MTCX Command Data 1	Kommandodaten: Byte 4 bis 7
0010h	MTCX Command Data 2	Kommandodaten: Byte 8 bis 11
0014h	MTCX Command Data 3	Kommandodaten: Byte 12 bis 15

Tabelle 9: MTCX Command Service Registerübersicht

Die einzelnen Register sind in den folgenden Kapiteln beschrieben.

5.3.1 MTCX Command Status Register

In diesem Register werden vom MTCX Status und Fehlercode für ein bearbeitetes Kommando übergeben.
Das Register liegt auf I/O Port 4110h.

MTCX Command Status Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	C	B			A							

ID	Name / Funktion	Default	Mode	Bemerkung
A	Error Code		RO	Fehlercode; nur gültig bei Status 011b: 00h = kein Fehler 80h = IRQ Kollision 81h = unbekanntes Kommando 82h = Timeout 83h = falscher Parameter 84h = Device busy 85h = Gerätefehler (DVI Data R/W) 86h = Datenfehler
B	Status		RO	Zustand des Auftrages: 000b = Schnittstelle ist frei 001b = Auftrag beendet ohne Fehler 010b = reserviert 011b = Auftrag beendet mit Fehler 100b bis 111b = reserviert
C	Lock	0	R/W	Sperrbit. Dieses Bit wird nicht vom MTCX ausgewertet und kann von der Anwendung für die Synchronisation mit anderen Aufträgen verwendet werden. 0 = Schnittstelle frei, 1 = gesperrt

Tabelle 10: MTCX Command Status Register

5.3.2 MTCX Command Param Register

In diesem Register müssen von der PC Anwendung die Parameter des vom MTCX zu bearbeitenden Kommandos eingetragen werden.
Das Register liegt auf I/O Port 4114h.

MTCX Command Param Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E								D				C				x	B	A													

ID	Name / Funk-tion	Default	Mode	Bemerkung
A	Addr		WO	Zieladresse.
B	Dir	0	WO	Kommandorichtung: 1 = Daten lesen 0 = Daten schreiben (bzw. programmieren)
C	DevNum	0	WO	Gerätenummer. Bei panelspezifischen Kommandos wird die Gerätenummer als Panel-nummer verwendet: 0 = 1. Gerät, 1 = 2. Gerät, ...15 = 16. Gerät Bei modulspezifischen Kommandos (Kommandocode 6xh) wird die Gerä-tenummer als Modulnummer verwendet: 0 = Systemeinheit 1 = Displayeinheit (nur PPC900) 2 = Buseinheit (optional auf PPC900) 3 = Display Link (optional, nur APC910) 4 = IF Option 1 (optional) [2] [3] 5 = IF Option 2 (optional) 6 = Speichermodul 1 (optional) 7 = Speichermodul 2 (optional) 8 = Lüfter Kit (optional) 9 = Slide-In 1 (optional) 10 = Slide-In 2 (optional, nur APC910) 11 = CPU Board 12 = Bus Lüfter Kit (optional, nur PPC900)
D	Target	0	WO	Gibt das Zielgerät an: 1 = Baseboard 3 = Panel 4 = Panel Settings 8 = USV und andere Werte (bei modulspezifischen Kommandos)
E	Command	0	WO	Kommandocode. Das Beschreiben dieses Bytes löst im MTCX den Interrupt für die Kom-mandobearbeitung aus

Tabelle 11: MTCX Command Param Register

Information:

Wenn Sie in Ihrer PC Anwendung keine 32 Bit Zugriffe auf I/O Ports durchführen können, dürfen Sie Byte 3 des Command Param Registers erst beschreiben, nachdem Bytes 0 bis 2 bereits mit gültigen Werten beschrieben wurden, weil dabei die Kommandobearbeitung am MTCX ausgelöst wird.

[2] Eine USV IF Option wird als IF Option 1 angesprochen.
[3] APC910: MTCX Versionen kleiner 0.07 verwenden vertauschte Modulnummern für die IF Optionen: 4 = IF Option 2 und 5 = IF Option 1.

5.3.3 MTCX Command Data [0..3] Register

Die MTCX Command Data 0 bis 3 Register beinhalten den Datenbereich für das Lesen und Schreiben von Daten und liegen ab I/O Port 4118h. Maximal können bis zu 16 Bytes übertragen werden. Alle Register haben denselben Aufbau.

MTCX Command Data [0..3] Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A																															

ID	Name / Funktion	Default	Mode	Bemerkung
A	Data		R/W	Daten sind abhängig vom Kommando

Tabelle 12: MTCX Command Data [0..3] Register

5.3.4 Kommandobearbeitung

Die Bearbeitung eines Kommandos über die MTCX Kommandoschnittstelle läuft nach folgendem Schema ab.

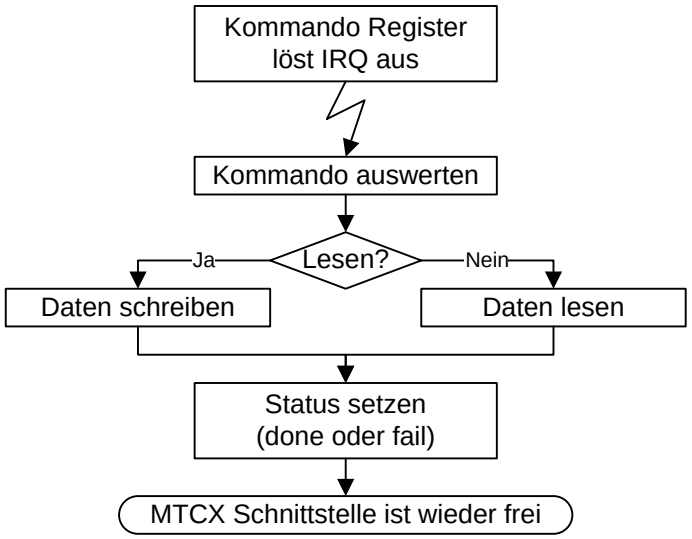

Abbildung 1: MTCX Kommandobearbeitung

Der MTCX ist bei der Kommandobearbeitung immer der Slave und der PC der Master. Die Kommandos werden von der PC Anwendung abgesetzt. Bei jedem Beschreiben des Command Param Register wird ein Interrupt im MTCX ausgelöst. In der Interrupt Funktion des MTCX wird das Kommando ausgewertet. Je nach Kommando werden dann Daten übernommen oder zurück geliefert. Am Ende der Kommandobearbeitung wird der Status im Command Status Register gesetzt. Anschließend ist die Kommandoschnittstelle wieder frei.

Durch das Lock Bit im Command Status Register kann die PC Anwendung selber verwalten, ob die Kommandoschnittstelle belegt ist.

5.3.5 MTCX Kommandos

Diese Kommandos werden vom MTCX unterstützt.

Kommando	Kommando Code	Bemerkung
Version Info	10h	Firmwareversionen lesen
Device Info	12h	Geräteinformationen lesen
Key Layer	18h	Tastenebene einstellen
Key Service	19h	Tastenbearbeitung steuern
Flash / EEPROM Service	20h	Flashspeicher und EEPROM Daten lesen/schreiben
Statistics Info	30h	Statistikwerte lesen (nur Automation Panels)
LED Service 0	40h	LEDs der 1. Ebene schalten
LED Service 1	41h	LEDs der 2. Ebene schalten
LED Service 2	42h	LEDs der 3. Ebene schalten
LED Service 3	43h	LEDs der 4. Ebene schalten
Display Service	48h	Display konfigurieren
Watchdog Service	50h	Watchdog konfigurieren, Software Reset
Module Info	61h	Geräteinformationen eines PC Moduls lesen
Module Tempera-ture	64h	Temperaturwerte eines PC Moduls lesen
Module Statistics	65h	Statistikwerte eines PC Moduls lesen
Module Correction	66h	Korrekturdaten eines PC Moduls lesen/schreiben
Module Voltage	68h	Spannungswerte eines PC Moduls lesen
Module Header	6Ah	Firmware Header eines PC Moduls lesen
Module Fan	6Bh	Lüfterwerte eines PC Moduls lesen
UPS Service	90h	USV Kommandos, Shutdown

Tabelle 13: MTCX Kommandos

Eine genaue Beschreibung der einzelnen Kommandos folgt in den nächsten Kapiteln.

5.3.5.1 Version Info Kommando

Mit diesem Kommando wird die Versionsinformation des jeweiligen Geräts gelesen.

Kommandofunktion	Kommando Code	Zielgerät	Richtung	Gerätenummer	Adresse	Datenlänge	Bemerkung
MTCX Version Info	10h	1 (Baseboard)	1	1	-	4 Byte	Version der MTCX Firmware lesen (kann auch über das **Version Register** durchgeführt werden, siehe Seite 12)
Scancode Version Info	10h	1 (Baseboard)	1	2	-	4 Byte	Version des Scancode Datenformats lesen

Tabelle 14: Version Info Kommando

Die mit dem Version Info Kommando gelesenen Daten besitzen denselben Aufbau wie das **Version** Register des MTCX (siehe Seite 12).

5.3.5.2 Device Info Kommando

Mit diesem Kommando kann die User Serial ID gelesen und eingestellt und die Geräteinformation aus den Factory Settings des PC gelesen werden.

Information:

Auf einem APC910 wird mit diesem Kommando immer die Systemeinheit und auf einem PPC900 immer das CPU Board angesprochen! Die Informationen anderer PC Module können über das neue Module Info Kommando des MTCX (siehe Seite 29) gelesen werden.

Kommandofunktion	Kommando Code	Zielgerät	Richtung	Gerätenummer	Adresse	Datenlänge	Bemerkung
ID Service	12h	1 (Baseboard)	1/0	0	0	8 Byte	User Serial ID lesen/setzen
Manufacturer Info	12h	1 (Baseboard)	1	1	0	16 Byte	Gerätekennung, Herstellerkennung etc. lesen
Serial Number	12h	1 (Baseboard)	1	2	0	16 Byte	Seriennummer lesen
Model Number 0	12h	1 (Baseboard)	1	2	10h	16 Byte	Bestellnummer Teil 1 lesen
Model Number 1	12h	1 (Baseboard)	1	2	20h	16 Byte	Bestellnummer Teil 2 lesen
Model Number 2	12h	1 (Baseboard)	1	2	30h	9 Byte	Bestellnummer Teil 3 lesen
Parent Info	12h	1 (Baseboard)	1	2	40h	8 Byte	Parent Gerätekennung und Kompatibilitätskennung lesen

Tabelle 15: Device Info Kommando

Die folgende Tabelle beschreibt die **ID Service** Daten.

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Reserve	4 Byte	
04h	User	4 Byte	Kann vom Kunden frei definiert werden: 00000000h bis FFFFFFFFh Wird im eingebauten EEPROM gespeichert.

Tabelle 16: ID Service Daten

Hinweis: Eine geänderte User Serial ID wird vom MTCX sofort übernommen – es ist kein Neustart des Geräts notwendig.

Die folgende Tabelle beschreibt die **Manufacturer Info** Daten:

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Device ID	4 Byte	Gerätekennung (SAP Materialnummer): 00000000h bis FFFFFFFFh
04h	Compatibility ID	2 Byte	Kompatibilitätskennung: 0000h bis FFFFh
06h	Vendor ID	4 Byte	Herstellerkennung: 0 = B&R
0Ah	Hardware Revision	5 Byte	Hardwarerevision als ASCII String (inkl. Null-Zeichen)
0Fh	Reserve	1 Byte	

Tabelle 17: Manufacturer Info Daten

Die folgende Tabelle beschreibt die **Serial Number** Daten:

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Serial Number	12 Byte	Seriennummer des PC als ASCII String (inkl. Null-Zeichen)
0Ch	Reserve	4 Byte	

Tabelle 18: Serial Number Daten

Die folgenden Tabellen beschreiben die **Model Number** Daten:

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Model Number Part 1	16 Byte	Teil 1 (Byte 0 bis 15) der Bestellnummer als ASCII String (inkl. Null-Zeichen)

Tabelle 19: Model Number 0 Daten

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Model Number Part 2	16 Byte	Teil 2 (Byte 16 bis 31) der Bestellnummer als ASCII String (inkl. Null-Zeichen)

Tabelle 20: Model Number 1 Daten

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Model Number Part 3	9 Byte	Teil 3 (Byte 32 bis 41) der Bestellnummer als ASCII String (inkl.s Null-Zeichen)

Tabelle 21: Model Number 2 Daten

Die folgende Tabelle beschreibt die **Parent Info** Daten:

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Device ID	4 Byte	Gerätekennung (SAP Materialnummer) des Parent Geräts: 00000000h bis FFFFFFFFh (= kein Parent)
04h	Compatibility ID	2 Byte	Kompatibilitätskennung (binär) des Parent Geräts: 0000h bis FFFFh (= kein Parent)
06h	Reserve	2 Byte	

Tabelle 22: Parent Info Daten

5.3.5.3 Key Layer Kommando

Mit diesem Kommando wird die aktuelle Tastenebene eingestellt.

Information:

Die Tastenebene wird nur eingestellt, wenn eine gültige Tastenkonfiguration auf dem Gerät vorhanden ist - das Kommando liefert aber keinen Fehler! Der Status der Tastenkonfiguration kann durch Lesen des Baseboard KeyState Register des MTCX (siehe Seite 44) ermittelt werden.

Kommandofunktion	Kommando Code	Zielgerät	Richtung	Gerätenummer	Adresse	Datenlänge	Bemerkung
Key Layer	18h	1 (Baseboard)	0	0	-	4 Byte	Tastenebene einstellen

Tabelle 23: Key Layer Kommando

Die folgende Tabelle beschreibt **die Key Layer** Daten.

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Key Layer	1 Byte	Tastenebene: 0 bis 3
01h	Reserve	3 Byte	

Tabelle 24: Key Layer Daten

Hinweis: Die aktuell eingestellte Tastenebene kann vom **Baseboard Key State** Register des MTCX (siehe Seite 44) gelesen werden.

5.3.5.4 Key Service Kommando

Mit diesem Kommando kann die Bearbeitung der Tasten (Scancodes, Panelsperre) der Displayeinheit eines PPC900 und eines angeschlossenen Automation Panel gesteuert werden.

Kommandofunktion	Kommando Code	Zielgerät	Richtung	Gerätenummer	Adresse	Datenlänge	Bemerkung
Scancode Lock	19h	1 (Baseboard)	1/0	0	-	4 Byte	Scancodes eines Panels sperren
Scancode Restart	19h	1 (Baseboard)	1	1	-	-	Scancodes neu einlesen (z.B. nach Update)
Panel Lock Time	19h	1 (Baseboard)	1/0	3	-	4 Byte	Panelsperrzeit lesen/setzen
Panel Lock	19h	1 (Baseboard)	1/0	4	-	4 Byte	Panels sperren/freigeben

Tabelle 25: Key Service Kommando

Hinweis: Scancode Lock und Scancode Restart sind nur sinnvoll, wenn eine gültige Tastenkonfiguration auf dem Gerät vorhanden ist.

Die folgende Tabelle beschreibt die **Scancode Lock** Daten.

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Lock Bits	2 Byte	Sperrbits: Bit x = 1: Scancodes von Panel x sind gesperrt Bit x = 0: Scancodes von Panel x sind freigegeben
02h	Reserve	2 Byte	

Tabelle 26: Scancode Lock Daten

Hinweis: Ob die Scancodes eines Panels gesperrt sind, kann auch direkt vom **Panel Flags** Register des MTCX (siehe Seite 53) gelesen werden.

Die folgende Tabelle beschreibt die **Panel Lock Time** Daten.

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Lock Time	2 Byte	Sperrzeit der Panels in ms: 0 bis 65535
02h	Reserve	2 Byte	

Tabelle 27: Panel Lock Time Daten

Die Panelsperrzeit gibt an, wie lange die Eingaben eines Panels (über resistiven Touchscreen und Matrixtasten) gesperrt werden, wenn auf einem anderen Panel gerade Eingaben erfolgen.

Die Panelsperrzeit wird bei Neustart des PC oder neuerlichem Einlesen der Scancodes (z.B. nach Download einer neuen Tastenkonfiguration) wieder auf den in der Tastenkonfiguration parametrierten Wert eingestellt.

Die folgende Tabelle beschreibt die **Panel Lock** Daten beim Lesen (Richtung 1).

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Lock Bits	2 Byte	Sperrbits: Bit x = 1: Panel x ist gesperrt Bit x = 0: Panel x ist freigegeben
02h	Reserve	2 Byte	

Tabelle 28: Panel Lock Daten (Lesen)

Die folgende Tabelle beschreibt die **Panel Lock** Daten beim Schreiben (Richtung = 0).

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Lock Bits	2 Byte	Sperrbits: Bit x = 1: Panel x ist gesperrt Bit x = 0: Panel x ist freigegeben
02h	Mask Bits	2 Byte	Maskierung der Sperrbits: Bit x = 1: Lock Bit x auswerten, Bit x = 0: Lock Bit x ignorieren

Tabelle 29: Panel Lock Daten (Schreiben)

Panels werden normalerweise automatisch gesperrt, wenn auf einem anderen Panel Eingaben erfolgen. Mit diesem Kommando können Panels gezielt gesperrt werden.

Hinweis: Ob ein Panel gesperrt ist, kann auch direkt vom **Panel Flags** Register des MTCX (siehe Seite 53) gelesen werden.

5.3.5.5 Flash / EEPROM Service Kommando

Mit diesem Kommando werden Flashspeicher und EEPROM Daten gelesen und programmiert.

Kommandofunktion	Kommando Code	Zielgerät	Richtung	Gerätenummer	Adresse	Datenlänge	Bemerkung
Scancodes	20h	1 (Baseboard)	1/0	2	0 bis 3FFFh	16 Byte	Tastenkonfiguration lesen/schreiben Max. 256 KBytes (Flashspeicher)
Factory Settings	20h	1 (Baseboard)	1/0	3	0 bis 7Fh	16 Byte	Factory Settings (der Systemeinheit) lesen/schreiben Max. 2048 Bytes (EEPROM)
Panel FPGA	20h	3 (Panel)	1/0	0 bis 14	0 bis 3FFFh[4]	16 Byte	Panel FPGA Firmware lesen/schreiben Max. 256 Kbytes (Flashspeicher) auf AP900 und AP800 bzw. max. 640 KBytes auf AP830 und AP9x3
Panel Factory Settings	20h	4 (Panel Settings)	1/0	0 bis 14	0 bis Fh bzw. 7Fh	16 Byte	Panel Factory Settings lesen/schreiben 256 Bytes (EEPROM) auf AP900 und AP800 bzw. max. 2048 Bytes auf AP830 und AP9x3

Tabelle 30: Flash / EEPROM Service Kommando

In den Datenregistern werden pro Kommandoaufruf bis zu 16 Bytes Daten bzw. Firmwarecode abgelegt.

Vorsicht!

Stellen Sie sicher, dass Sie auf die Flash und EEPROM Daten nur lesend zugreifen. Andernfalls kann es zum Überschreiben von Daten oder Firmware kommen, die für den Betrieb des PC notwendig sind!

Information:

Die „Panel FPGA“ und „Panel Factory Settings“ Kommandofunktion werden von der Displayeinheit eines PPC900 (Panel 15) und von AP800 Extension Units nicht unterstützt. Ob es sich um eine AP800 Extension Unit handelt, kann über das Panel Version Register (siehe Seite 51) ermittelt werden.

[4] Auf AP830 und AP9x3 müssen für den Zugriff auf die Panel Firmware spezielle Adresswerte verwendet werden (nur für B&R interne Verwendung).

5.3.5.6 Statistics Info Kommando

Mit diesem Kommando werden die Statistikwerte von angeschlossenen Automation Panels gelesen.

Information:

Das Kommando wird von der Displayeinheit eines PPC900 (Panel 15) und AP800 Extension Units nicht unterstützt. Ob es sich um eine AP800 Extension Unit handelt, kann über das Panel Version Register (siehe Seite 51) ermittelt werden.
Die Statistikwerte der PC Module können mit dem neuen Module Statistics Kommando des MTCX (siehe Seite 32) gelesen werden.

Kommandofunktion	Kommando Code	Zielgerät	Richtung	Gerätenummer	Adresse	Datenlänge	Bemerkung
Panel Statistics	30h	3 (Panel)	1	0 bis 14	-	4 Byte	Betriebsstunden und Einschaltzyklen (der Hintergrundbeleuchtung) eines Panels lesen

Tabelle 31: Statistics Info Kommando

Die folgende Tabelle beschreibt die **Panel Statistics** Daten.

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Power-on Cycles	2 Byte	Einschaltzyklen der Hintergrundbeleuchtung: 0 bis 65355
02h	Power-on Hours	2 Byte	Betriebsstunden der Hintergrundbeleuchtung: 0 bis 65535

Tabelle 32: Panel Statistics Daten

5.3.5.7 LED Service Kommando

Mit diesem Kommando können die LEDs der Displayeinheit eines PPC900 und eines angeschlossenen Automation Panels gelesen oder gesetzt werden.

Kommandofunktion	Kommando Code	Zielgerät	Richtung	Gerätenummer	Adresse	Datenlänge	Bemerkung
LED Service 0	40h	3 (Panel)	1/0	0 bis 15	[5]	16 Byte	64 LEDs der 1. Ebene lesen/setzen
LED Service 1	41h	3 (Panel)	1/0	0 bis 15	[5]	16 Byte	64 LEDs der 2. Ebene lesen/setzen
LED Service 2	42h	3 (Panel)	1/0	0 bis 15	[5]	16 Byte	64 LEDs der 3. Ebene lesen/setzen
LED Service 3	43h	3 (Panel)	1/0	0 bis 15	[5]	16 Byte	64 LEDs der 4. Ebene lesen/setzen

Tabelle 33: LED Service Kommando

Es können mit einem Kommando maximal 64 LEDs gleichzeitig gelesen oder gesetzt werden. Zum Lesen/Setzen aller LEDs sind daher zwei Kommandos notwendig. Die Anzahl der LEDs wird vom Anwender angegeben. Zum Beispiel ist es möglich, ab der 14. LED nur 3 LEDs einer Ebene zu lesen oder zu setzen.

Information:

Die LED Service Kommandos liefern keinen Fehler, wenn das Gerät gar keine LEDs unterstützt. Dieser Fall muss in der Anwendung abgefangen werden, z.B. durch vorheriges Lesen der LED Anzahl vom Panel Specials Register des MTCX (siehe Seite 52).

Die folgende Tabelle beschreibt die **LED Service** Daten.

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	LED[0..15] bzw. [64..79]	4 Byte	2 Bits pro LED Zustand
04h	LED[16..31] bzw. [80..95]	4 Byte	2 Bits pro LED Zustand
08h	LED[32..47] bzw. [96..111]	4 Byte	2 Bits pro LED Zustand
0Ch	LED[48..63] bzw. [112..127]	4 Byte	2 Bits pro LED Zustand

Tabelle 34: LED Service Daten

Jeder LED Zustand wird mit 2 Bits definiert. Folgende Bitmuster sind gültig.

Bits	LED Zustand
00b	Aus
01b	Langsam Blinken
10b	Schnell Blinken
11b	Ein

Tabelle 35: LED Zustände

[5] Adresse: Bit 0..7 = LED Offset, Bit 8..13 = Anzahl der LEDs – 1

5.3.5.8 Display Service Kommando

Mit diesem Kommando werden Einstellungen der Displayeinheit eines PPC900 und eines angeschlossenen Automation Panel gelesen und geändert.

Kommandofunktion	Kommando Code	Zielgerät	Richtung	Gerätenummer	Adresse	Datenlänge	Bemerkung
Display Settings	48h	3 (Panel)	1/0	0 bis 15	0	4 Byte	Einstellungen des Displays lesen/schreiben.

Tabelle 36: Display Service Kommando

Die folgende Tabelle beschreibt die **Display Settings** Daten.

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Brightness	1 Byte	Helligkeit der Hintergrundbeleuchtung in Prozent: 0 bis 100, 255 = nicht unterstützt
01h	Reserve	3 Byte	

Tabelle 37: Display Settings Daten

5.3.5.9 Watchdog Service Kommando

Mit diesem Kommando wird der Watchdog konfiguriert und ein Reset des PC per Software durchgeführt.

Kommandofunktion	Kommando Code	Zielgerät	Richtung	Gerätenummer	Adresse	Datenlänge	Bemerkung
Watchdog Config	50h	1 (Baseboard)	1/0	0	-	4 Byte	Watchdog Zeitfenster lesen/einstellen
Software Reset	50h	1 (Baseboard)	0	2	-	-	Reset des PC durchführen

Tabelle 38: Watchdog Service Kommando

Die folgende Tabelle beschreibt die **Watchdog Config** Daten.

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Minimum Time	2 Byte	Watchdog Minimalzeit: 0 bis 65535 ms
02h	Maximum Time	2 Byte	Watchdog Maximalzeit: 0 bis 65535 ms

Tabelle 39: Watchdog Config Daten

Hinweise:

- Wenn die Minimum Time größer als die Maximum Time ist, werden die Zeiten vom MTCX automatisch getauscht: die Minimum Time wird die Maximum Time und umgekehrt.
- Wenn Minimum Time und Maximum Time denselben Wert (größer 0) haben, wird vom MTCX ein Fehler geliefert.
- Wenn Minimum Time und Maximum Time 0 sind, wird der Watchdog deaktiviert.
- Nachdem das Zeitfenster des Watchdog eingestellt wurde, kann der Watchdog durch erstmaliges Beschreiben des Toggle Bit im Baseboard Support Register (siehe Seite 43) aktiviert werden.

5.3.5.10 Module Info Kommando

Mit diesem Kommando werden Geräteinformationen eines PC Moduls gelesen.

Kommandofunktion	Kommando Code	Zielgerät	Richtung	Gerätenummer	Adresse	Datenlänge	Bemerkung
Module Device ID	61h	0	1	0 bis 12 (Modulnummer)	100h	4 Byte	Gerätekennung eines Moduls lesen
Module Vendor & Compatibility ID	61h	0	1	0 bis 12 (Modulnummer)	200h	6 Byte	Hersteller- und Kompatibilitätskennung eines Moduls lesen
Module HW Revision	61h	0	1	0 bis 12 (Modulnummer)	300h	5 Byte	HW Revision eines Moduls lesen
Module Serial Number	61h	0	1	0 bis 12 (Modulnummer)	400h	12 Byte	Seriennummer eines Moduls lesen
Module Model Number 0	61h	0	1	0 bis 12 (Modulnummer)	500h	16 Byte	Bestellnummer Teil 1 eines Moduls lesen
Module Model Number 1	61h	0	1	0 bis 12 (Modulnummer)	501h	16 Byte	Bestellnummer Teil 2 eines Moduls lesen
Module Model Number 2	61h	0	1	0 bis 12 (Modulnummer)	502h	9 Byte	Bestellnummer Teil 3 eines Moduls-lesen
Module Parent Info	61h	0	1	0 bis 12 (Modulnummer)	700h	6 Byte	Parent Gerätekennung und Kompatibilitätskennung eines Moduls lesen

Tabelle 40: Module Info Kommando

Jedes PC Modul kann theoretisch Geräteinformationen besitzen. Das Kommando liefert für nicht vorhandene Geräteinformationen einen Fehler.

Hinweise:

- Auf APC910 liefert das Modul 11 (CPU Board) eingeschränkte Geräteinformationen, d.h. nur Device ID, Hardware Revision und Model Number sind mit gültigen Werten versorgt – die Serial Number z.B. ist „leer“.
- Auf PPC900 werden für Modul 0 (Systemeinheit) und Modul 11 (CPU Board) dieselben Geräteinformationen geliefert.
- Auf PPC900 werden für die Module 6 und 7 (Speichermodul 1 und 2) die Seriennummer und Bestellnummer aus den SPD Daten des DRAMs und für die anderen Geräteinformationen Pseudo-Werte geliefert.

Die folgende Tabelle beschreibt die **Module Device ID** Daten:

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Device ID	4 Byte	Gerätekennung (SAP Materialnummer): 00000000h bis FFFFFFFFh

Tabelle 41: Module Device ID Daten

Die folgende Tabelle beschreibt die **Module Vendor & Compatibility ID** Daten:

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Vendor ID	4 Byte	Herstellerkennung: 0 = B&R
04h	Compatibility ID	2 Byte	Kompatibilitätskennung: 0000h bis FFFFh

Tabelle 42: Module Vendor & Compatibility ID Daten

Die folgende Tabelle beschreibt die **Module HW Revision** Daten:

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Hardware Revision	5 Byte	Hardwarerevision als ASCII String (inkl. Null-Zeichen)

Tabelle 43: Module HW Revision Daten

Die folgende Tabelle beschreibt die **Module Serial Number** Daten:

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Serial Number	12 Byte	Seriennummer als ASCII String (inkl. Null-Zeichen)

Tabelle 44: Module Serial Number Daten

Die folgenden Tabellen beschreiben die **Module Model Number** Daten:

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Model Number Part 1	16 Byte	Teil 1 (Byte 0 bis 15) der Bestellnummer als ASCII String (inkl. Null-Zeichen)

Tabelle 45: Module Model Number 0 Daten

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Model Number Part 2	16 Byte	Teil 2 (Byte 16 bis 31) der Bestellnummer als ASCII String (inkl. Null-Zeichen)

Tabelle 46: Module Model Number 1 Daten

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Model Number Part 3	9 Byte	Teil 3 (Byte 32 bis 41) der Bestellnummer als ASCII String (inkl. Null-Zeichen)

Tabelle 47: Module Model Number 2 Daten

Die folgende Tabelle beschreibt die **Module Parent Info** Daten:

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Device ID	4 Byte	Gerätekennung (SAP Materialnummer) des Parent Geräts: 00000000h bis FFFFFFFFh (= kein Parent)
04h	Compatibility ID	2 Byte	Kompatibilitätskennung (binär) des Parent Geräts: 0000h bis FFFFh (= kein Parent)

Tabelle 48: Module Parent Info Daten

5.3.5.11 Module Temperature Kommando

Mit diesem Kommando werden die Temperaturwerte eines PC Moduls gelesen.

Kommandofunktion	Kommando Code	Zielgerät	Richtung	Gerätenummer	Adresse	Datenlänge	Bemerkung
Module Temperature	64h	0 bis 3 (Sensornummer)	1	0, 1, 3, 6, 7, 9, 10 und 11 (Modulnummer)	0	4 Byte	Temperatursensorwert eines Moduls lesen

Tabelle 49: Module Temperature Kommando

Jedes PC Modul kann theoretisch bis zu vier Temperatursensoren besitzen. Das Kommando liefert für nicht vorhandene Temperatursensoren einen Fehler.

Hinweis: Aktuell können Temperaturwerte nur für Modul 0, 1, 3, 6, 7, 9, 10 und 11 gelesen werden (6 und 7 nur auf PPC900).

Die folgende Tabelle beschreibt die **Module Temperature** Daten:

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Status	1 Byte	B&R intern
01h	Reserve	1 Byte	
02h	Value	2 Byte	Bit 0 bis 5: reserviert Bit 6 bis 15: Temperaturwert in 0.25 °C

Tabelle 50: Module Temperature Daten

5.3.5.12 Module Statistics Kommando

Mit diesem Kommando werden Statistikwerte eines PC Moduls gelesen.

Kommandofunktion	Kommando Code	Zielgerät	Richtung	Gerätenummer	Adresse	Datenlänge	Bemerkung
Module Statistics	65h	0	1	0, 1, 2, 3, 4, 5 (Modulnummer)	100h	8 Byte	Betriebsstunden und Einschaltzyklen eines Moduls lesen.
Display Statistics	65h	1	1	1 (Modulnummer)	100h	8 Byte	Betriebsstunden und Einschaltzyklen der Hintergrundbeleuchtung der Displayeinheit lesen.
Fan Statistics	65h	0 bis 3 (Lüfternummer)	1	8, 12 (Modulnummer)	200h	8 Byte	Betriebsstunden und Einschaltzyklen der Lüfter lesen.
UPS Statistics	65h	3	1	4 (Modulnummer)	100h	8 Byte	„On Battery“ Stunden und Zyklen der USV lesen.

Tabelle 51: Module Statistics Kommando

Jedes PC Modul kann theoretisch Statistikwerte besitzen. Das Kommando liefert für nicht vorhandene Statistikwerte einen Fehler.

Hinweis: Aktuell können Statistikwerte nur für Modul 0, 1, 2, 3, 4, 5, 8 und 12 gelesen werden (1 und 12 nur auf PPC900).

Die folgende Tabelle beschreibt die **Module Statistics** Daten.

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Power-on Hours	4 Byte	Betriebsstunden (in ¼ Stunden): 0 bis 16777212
04h	Power-on Cycles	4 Byte	Einschaltzyklen: 0 bis 16777212

Tabelle 52: Module Statistics Daten

Die folgende Tabelle beschreibt die **Display Statistics** Daten.

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Backlight-on Hours	4 Byte	Betriebsstunden (in ¼ Stunden): 0 bis 16777212
04h	Backlight-on Cycles	4 Byte	Einschaltzyklen: 0 bis 16777212

Tabelle 53: Display Statistics Daten

Die folgende Tabelle beschreibt die **Fan Statistics** Daten.

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Fan-on Hours	4 Byte	Betriebsstunden (in ¼ Stunden): 0 bis 16777212
04h	Fan-on Cycles	4 Byte	Einschaltzyklen: 0 bis 16777212

Tabelle 54: Fan Statistics Daten

Die folgende Tabelle beschreibt die **UPS Statistics** Daten.

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	„On Battery“ Hours	4 Byte	USV „On Battery“ Stunden (in ¼ Stunden): 0 bis 16777212
04h	„On Battery“ Cycles	4 Byte	USV „On Battery“ Zyklen: 0 bis 16777212

Tabelle 55: UPS Statistics Daten

5.3.5.13 Module Correction Kommando

Mit diesem Kommando werden Korrekturdaten eines PC Moduls gelesen.

Kommandofunktion	Kommando Code	Zielgerät	Richtung	Gerätenummer	Adresse	Datenlänge	Bemerkung
UPS User Settings	66h	0	1/0	4 (Modulnummer)	600h	8/4 Byte	Korrekturdaten eines Moduls lesen/schreiben

Tabelle 56: Module Correction Kommando

Jedes PC Modul kann theoretisch Korrekturdaten besitzen. Das Kommando liefert für nicht vorhandene Korrekturdaten einen Fehler.

Hinweis: In diesem Dokument sind nur die Korrekturdaten für die USV IF Option beschrieben.

Die **UPS User Settings** liegen im EEPROM der USV und beinhalten derzeit nur die Abschaltzeit der USV bei niedrigem Batteriestand (Low Battery Shutdown Time). Das ist die Zeit, die von der USV bei niedrigem Batteriepegel im Batteriebetrieb oder anderen Fehlern (z.B. Übertemperatur) gewartet wird, bevor sie die Stromversorgung abschaltet.

Damit wird verhindert, dass die USV Batterie zu sehr entladen wird, wenn die USV Dienst nicht vom Betriebssystem mit dem UPS Shutdown Kommando ausgeschalten wird (siehe Seite 39) bzw. das Betriebssystem nicht runterfährt.

Vorsicht!

Ein Low Battery Shutdown unterbricht ein eventuell bereits laufendes UPS Shutdown Kommando (siehe Seite 39) und kann damit die geplante Abschaltzeit des Systems verkürzen!

Die folgende Tabelle beschreibt die **UPS User Settings** Daten beim Lesen.

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Ok	1 Byte	„Daten Ok“ Flag: 0 oder 1
01h	Special	1 Byte	B&R intern
02h	Counter	2 Byte	Anzahl der Schreibvorgänge für diese Daten
04h	LowBatShutdownTime	2 Byte	Zeit in Sekunden, bis die USV nach einem „Low Battery“ Alarm ausschaltet
06h	Reserve	2 Byte	FFFFh

Tabelle 57: UPS User Settings Daten (lesen)

Information:

Wenn auf der USV noch keine UPS User Settings vorhanden sind, wird vom MTCX beim Lesen der Fehler 86h geliefert und der Standardwert 180 Sekunden als LowBatShutdownTime verwendet.

Die folgende Tabelle beschreibt die **UPS User Settings** Daten beim Schreiben.

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	LowBatShutdownTime	2 Byte	Zeit in Sekunden, bis die USV nach einem „Low Battery“ Alarm ausschaltet: 10 bis 1200. [6]
02h	Reserve	2 Byte	FFFFh

Tabelle 58: UPS User Settings Daten (schreiben)

Hinweis: Geänderte **UPS User Settings** werden sofort übernommen – es ist kein Neustart des Geräts notwendig.

[6] Werte außerhalb des gültigen Bereichs werden vom MTCX begrenzt.

5.3.5.14 Module Voltage Kommando

Mit diesem Kommando werden Spannungswerte eines PC Moduls gelesen.

Kommandofunktion	Kommando Code	Zielgerät	Richtung	Gerätenummer	Adresse	Datenlänge	Bemerkung
Voltage	68h	0	1	0 (Modulnummer)	0	8 Byte	Spannungswerte der Systemeinheit lesen

Tabelle 59: Module Voltage Kommando

Hinweis: Aktuell können Spannungswerte nur für Modul 0 gelesen werden.

Die folgende Tabelle beschreibt die **Voltage** Daten:

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Vbat	2 Byte	Batteriespannung: Bit 0 bis 14: Spannung in 10 mV. Bit 15: Status 0 = gut, 1 = schlecht
02h	Vin	2 Byte	Eingangsspannung: Bit 0 bis 14: Spannung in 100 mV. **Der Wert ist immer 0 auf APC910 und PPC900!** [7] Bit 15: Status 0 = gut, 1 = schlecht
04h	Wcpu	2 Byte	Leistung des gesamten Systems in 10 mW [8] [9]
06h	Reserve	2 Byte	

Tabelle 60: CPU Voltage Daten

[7] Benötigt MTCX Version 1.01 oder höher auf APC910 und MTCX Version 1.02 oder höher auf PPC900. Ältere MTCX Versionen liefern ungültige Werte.
[8] Auf anderen Systemen gibt dieser Wert nur die Leistung der CPU an.
[9] Benötigt MTCX Version 1.01 oder höher auf APC910 und MTCX Version 1.02 oder höher auf PPC900. Ältere MTCX Versionen liefern ungültige Werte.

5.3.5.15 Module Header Kommando

Mit diesem Kommando wird der Header der aktuell gebooteten FPGA Firmware eines PC Moduls gelesen.

Kommandofunktion	Kommando Code	Zielgerät	Richtung	Gerätenummer	Adresse	Datenlänge	Bemerkung
FPGA Header	6Ah	0	1	0, 3 (Modulnummer)	0	16 Byte	FPGA Header lesen

Tabelle 61: Module Header Kommando

Hinweis: Aktuell kann der Header nur für Modul 0 und 3 gelesen werden.

Die folgende Tabelle beschreibt die **FPGA Header** Daten:

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	ID	4 Byte	Headerkennung: „APC9“ auf Systemeinheit (Modul 0) „LSDL“ auf SDL Display Link (Modul 3) „P9S3“ auf SDL3 Display Link (Modul 3)
04h	Version	4 Byte	FPGA Firmware Version als ASCII String (ohne Null-Zeichen): „vvyy“. vv ist die Major Version und yy die Minor Version.
08h	Counter	4 Byte	Updatezähler: 0 bis 4294967295. Wird bei jedem Update um 1 erhöht.
0Ch	Reserve	2 Byte	
0Eh	CRC	2 Byte	16 Bit CRC für Header.

Tabelle 62: FPGA Header Daten

5.3.5.16 Module Fan Kommando

Mit diesem Kommando werden die Lüfterdrehzahlen eines PC Moduls gelesen.

Kommandofunktion	Kommando Code	Zielgerät	Richtung	Gerätenummer	Adresse	Datenlänge	Bemerkung
Module Fan	6Bh	0 bis 3 (Lüfternummer)	1	8, 12 (Modulnummer)	0	4 Byte	Lüfterdrehzahl des Lüfter Kit Moduls lesen

Tabelle 63: Module Fan Kommando

Jedes PC Modul kann theoretisch bis zu vier Lüfter besitzen. Das Kommando liefert für nicht vorhandene Lüfter einen Fehler.

Hinweis: Aktuell können Lüfterdrehzahlen nur für Modul 8 und 12 (nur auf PPC900) gelesen werden.

Information:

Wenn kein Lüfter Kit vorhanden ist, können zwar Lüfterdrehzahlen gelesen werden – diese sind aber immer 0 und im Status ist Bit 1 gesetzt. Um zu erkennen, ob ein Lüfter Kit vorhanden ist, können Sie mit dem Module Info Kommando (siehe Seite 29) versuchen, die Device ID des Moduls zu lesen.

Die folgende Tabelle beschreibt die **Module Fan** Daten:

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Status	1 Byte	Lüfterstatus: Bit 0 = 1: Lüfter zu langsam, Unterschreitung der Mindestdrehzahl Bit 1 = 1: Lüfterstillstand Alarm
01h	Reserve	1 Byte	
02h	Value	2 Byte	Lüfterdrehzahl: 0 bis 65535 U/min

Tabelle 64: Module Fan Daten

5.3.5.17 UPS Service Kommando

Mit diesem Kommando können spezielle Funktionen der USV gesteuert werden.

Kommandofunktion	Kommando Code	Zielgerät	Richtung	Gerätenummer	Adresse	Datenlänge	Bemerkung
UPS Shutdown	90h	8 (UPS)	0	0	1	4 Byte	USV abschalten

Tabelle 65: UPS Service Kommando

Die folgende Tabelle beschreibt die **UPS Shutdown** Daten.

Offset	Name / Funktion	Datenlänge	Bemerkung
00h	Delay	2 Byte	Verzögerungszeit für Abschalten der USV in Sekunden: 10 bis 1200 [10]
02h	Reserve	2 Byte	

Tabelle 66: UPS Shutdown Daten

Dieses Kommando kann vom Betriebssystem bei einem Ausfall der Stromversorgung an die USV gesendet werden, bevor das Betriebssystem selbst herunterfährt.

Vorsicht!

Die angegebene Verzögerungszeit muss länger sein als die Zeit, die vom Betriebssystem zum Runterfahren benötigt wird. Andernfalls kann es zu Datenverlusten kommen. Beachten Sie dabei auch die Low Battery Shutdown Zeit der USV (siehe Seite 34)!

Information:

- **Die USV wird immer automatisch abgeschaltet, sobald das System in den Standby Zustand wechselt. Anm.: Die USV Batterie wird im Standby geladen. [11]**
  Die USV wird im Batteriebetrieb auch automatisch mit einer einstellbaren Zeit (siehe Seite 34) ausgeschaltet, wenn ein zu niedriger Batteriestand erkannt wird.
  Durch das UPS Shutdown Kommando wird die USV aber auch abgeschaltet, wenn das Betriebssystem nicht ordnungsgemäß herunterfährt. Dadurch wird die Batterie nicht weiter entladen.
- **Das UPS Shutdown Kommando ist auch notwendig, damit der PC automatisch neu startet, wenn während des Herunterfahrens vom System die Stromversorgung wieder funktioniert. [12]**
- **Das UPS Shutdown Kommando liefert keinen Fehler, wenn keine USV vorhanden ist. Dieser Fall muss in der Anwendung abgefangen werden, z.B. durch vorheriges Lesen des ked“ Flag im UPS Status Register (siehe Seite 47).**

[10] Werte außerhalb des gültigen Bereichs werden vom MTCX begrenzt.
[11] Das Laden im Standby wird erst ab MTCX Version 1.00 oder höher unterstützt.
[12] Auf APC910 ist dazu MTCX Firmware 1.00 oder höher und CPU Board Controller Firmware 9.32 oder höher erforderlich.

5.4 Hardware Info Register

Die Hardware Info Register beinhalten gerätespezifische Informationen und liegen ab I/O Port 4128h.

4128h+ Offset	Register	Beschreibung
0000h	Hardware Info	gerätespezifische Informationen
0004h	Hardware Info 2	Weitere gerätespezifische Informationen

Tabelle 67: MTCX Hardware Info Register Übersicht

Die einzelnen Register sind in den folgenden Kapiteln beschrieben.

5.4.1 Hardware Info Register

Dieses Register beinhaltet gerätespezifische Informationen und liegt auf I/O Port 4128h.

Hardware Info Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	X	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	D	C	B	A							

ID	Name / Funk-tion	Default	Mode	Bemerkung
A	Device Type		RO	Gerätetyp: 0Eh =APC910 0Fh = PPC900 00h und FFh bedeuten, dass der Gerätetyp nicht gelesen werden kann bzw. die Erkennung nicht unterstützt wird. Alle anderen Werte sind reserviert.
B	MTCX Boot Area		RO	MTCX (auf Systemeinheit) wurde gebootet vom: 0 = unteren Bereich 1 = oberen Bereich
C	BIOS Boot Area		RO	BIOS wurde gebootet vom: [13] 0 = Backupbereich 1 = normalen Bereich
D	FPGA Boot Area		RO	FPGA (des Display Link) wurde gebootet vom: 0 = unteren Bereich 1 = oberen Bereich

Tabelle 68: MTCX Hardware Info Register

[13] Nicht relevant auf APC910.

5.4.2 Hardware Info 2 Register

Dieses Register beinhaltet weitere gerätespezifische Informationen und liegt auf I/O Port 412Ch.

MTCX Hardware Info 2 Register																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A																															

ID	Name / Funktion	Default	Mode	Bemerkung
A	MTCX Counter		R/W	Updatezähler des aktuellen MTCX Image der Systemeinheit. Dieser Wert ist im INTEL Format und wird beim Booten einmal beschrieben.

Tabelle 69: MTCX Hardware Info 2 Register

5.5 Baseboard Service Register

Die Baseboard Service Register beinhalten Daten und Einstellungen des Baseboard und liegen ab I/O Port 4130h.

4130h+ Offset	Register	Beschreibung
0000h	Baseboard Support	NMI Status und Watchdog Toggle
0004h	Reserviert	
0008h	Reserviert	
000Ch	Reserviert	
0010h	Reserviert	
0014h	Reserviert	
0018h	Baseboard KeyState	Einstellungen der Tastenbearbeitung
001Ch	Baseboard Specials	Spezielle Einstellungen des Geräts

Tabelle 70: MTCX Baseboard Service Register Übersicht

Die einzelnen Register sind in den folgenden Kapiteln beschrieben.

5.5.1 Baseboard Support Register

Dieses Register beinhaltet das Watchdog Toggle Bit sowie Einstellungen für die SMC NMI Bearbeitung und liegt auf I/O Port 4130h.

Baseboard Support Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	x	x	x	D	C	B	A						E							x	x	x	x	x	x	x	x	x

ID	Name / Funktion	Default	Mode	Bemerkung
A	SMC Status	0	RO	Zeigt die Reset Ursache an: Bit 0 = 1: Watchdog Bit 1 = 1: Power Fail Bit 2 = 1: Reset Taster Bit 3 = 1: Software Reset Bit 4 = 1: Power Button
B	SMC NMI IRQ		RO	Zeigt an, ob der SMC NMI IRQ aktiv ist (kann in einer NMI Routine verwendet werden, um abzufragen, ob der NMI vom SMC ausgelöst wurde): 1 = aktiv 0 = inaktiv
C	SMC Status Invalid		RO	Zeigt an, ob der SMC Status gültig ist: 1 = Kein SMC Status zu lesen 0 = SMC Status ist gültig Der SMC Status ist beim Aufruf des NMI bereits gültig und bleibt nach dem Warmstart erhalten.
D	Watchdog Toggle		R/W	Watchdog Toggle Bit: muss zum Bestätigen des Watchdog mit 0 beschrieben werden.
E	NMI Skip Reset	0	WO	NMI Reset Skip: Nach jedem NMI Logik Ereignis kann man innerhalb der NMI Reset Zeit (Default 10 ms) den Reset für folgende Ursachen unterbinden. Bit 0 = 1: Watchdog Bit 2 = 1: Reset Taster Bit 3 = 1: Software Reset

Tabelle 71: MTCX Baseboard Support Register

5.5.2 Baseboard KeyState Register

Dieses Register beinhaltet spezielle Einstellungen für die Behandlung der Matrixtasten und Tastenkonfiguration und liegt auf I/O Port 4148h.

Baseboard KeyState Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	D	C								B				A			

ID	Name / Funktion	Default	Mode	Bemerkung
A	Level	0	R/W	Aktuell eingestellte Tastenebene: 0 bis 3
B	Mode	0	RO	Aktuell eingestellter Ebenenmodus: 0 = Shift, 1 = Toggle, 2 = One-Shot
C	Checksum	0	RO	Prüfsumme der Tastenkonfiguration im Flashspeicher
D	Status	0	RO	Zeigt an, ob die Tastenkonfiguration im Flashspeicher gültig ist: 0 = ungültig 1 = gültig

Tabelle 72: MTCX Baseboard KeyState Register

Information:

Eine mit dem Key Layer Kommando (siehe Seite 21) geänderte Tastenebene wird bedingt durch den internen Ablauf im MTCX erst nach kurzer Verzögerung in „Level“ im Baseboard Key State Register angezeigt.

5.5.3 Baseboard Specials Register

Dieses Register beinhaltet spezielle Werte des PC und liegt auf I/O Port 414Ch.

Baseboard Specials Register																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	x	X	x	x	x	x	x	x	x	x	x	x	x	x	x	x	C	B	A		x	x	x	x	x	x	x	x

ID	Name / Funk-tion	Default	Mode	Bemerkung
A	Bat State		RO	Zustand der CMOS Batterie: [14] 0 = Batterie ist gut 1 = reserviert 2 = reserviert 3 = Batterie ist schlecht
B	Power Button		RO	Status des Power Button: 0 = gedrückt 1 = nicht gedrückt
C	Reset Button		RO	Status des Reset Button: 0 = gedrückt 1 = nicht gedrückt

Tabelle 73: MTCX Baseboard Specials Register

[14] Die Batterie wird beim Start des PC und danach alle 24 Stunden kontrolliert. Bei der Messung wird die Batterie kurzzeitig (1 Sekunde) belastet.

5.6 UPS Service Register

Die UPS Service Register beinhalten Statusflags und Betriebsdaten der USV und liegen ab I/O Port 4150h.

4150h+ Offset	Register	Beschreibung
0000h	UPS Status	USV Statusflags und Batteriespannung
0004h	UPS Values	USV Betriebsdaten
0008h	UPS Specials	Spezielle USV Werte (nicht verwendet)

Tabelle 74: MTCX UPS Service Register Übersicht

Die einzelnen Register sind in den folgenden Kapiteln beschrieben.

5.6.1 UPS Status Register

Dieses Register beinhaltet Statuswerte der USV und liegt auf I/O Port 4150h.

UPS Status Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D								C																x	x	x	x	x	x	B	A

ID	Name / Funk-tion	Default	Mode	Bemerkung
A	Detected	0	RO	Zeigt an, ob die USV erkannt wurde: 1 = USV erkannt 0 = USV nicht erkannt
B	Linked	0	RO	Zeigt an, ob die USV verbunden ist: [15] 1 = USV verbunden 0 = USV nicht verbunden
C	Flags	0	RO	USV Statusflags: Bit 0 = 1: Stromversorgung in Ordnung Bit 1 = 1: Batterieeinstellungen nicht vorhanden / fehlerhaft [16] Bit 2 = 1: Batteriebetrieb Bit 3 = 1: Niedriger Batteriestand Bit 4 = 1: Batterie verpolt Bit 5 = 1: Batteriefehler (nicht vorhanden, falsche Spannung oder Tempe-ratur etc.) [17] Bit 6 = 1: Batterielebensdauer erreicht [18] Bit 7 = 1: Shutdown läuft [19] Bit 8 = 1: Überstromalarm Bit 9 = 1: Fabrikseinstellungen nicht vorhanden / fehlerhaft [20] Bit 10 = 1: Anwendereinstellungen nicht unterstützt [21] Bit 15 = 1: Batterietemperatur überschritten oder nicht messbar [22]
D	Bat Temp	0	RO	Temperatur der USV Batterie in °C: Messbereich ist -42 bis +82; -128 = Temperatur kann nicht gelesen werden

Tabelle 75: UPS Status Register

[15] „Linked“ zeigt an, ob die Kommunikation zwischen MTCX und USV funktioniert. Bei der USV IF Option sind „Detected“ und „Linked“ immer beide 0 oder 1.
[16] Wird vom MTCX nicht unterstützt (immer 0).
[17] Bit 5 kann alleine oder zusammen mit anderen Bits (z.B. Bit 4) gesetzt sein.
[18] Wird vom MTCX nicht unterstützt (immer 0).
[19] Entweder automatischer Shutdown von der USV selbst oder durch UPS Service Kommando (siehe Seite 39) ausgelöst.
[20] Ist immer 0. Anm.: Ohne gültige Fabrikseinstellungen wird die USV nicht erkannt („Detected“ und „Linked“ = 0).
[21] APC910: Anwendereinstellungen werden erst ab MTCX Version 0.08 oder höher unterstützt.
[22] Der Temperaturalarm wird bei -30 und +60 °C ausgelöst.

5.6.2 UPS Values Register

Dieses Register beinhaltet Betriebsdaten der USV und liegt auf I/O Port 4154h.

Information:

Das Register enthält nur relevante Daten, wenn die USV verbunden ist (siehe UPS Status Register auf Seite 47).

UPS Values Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B																A															

ID	Name / Funk-tion	Default	Mode	Bemerkung
A	Bat Current	0	RO	Pos. Werte: Ladestrom der USV Batterie in mA . Neg. Werte: Entladestrom der USV Batterie in mA.
B	Bat Voltage	0	RO	Spannung der USV Batterie in mV.

Tabelle 76: UPS Status2 Register

5.6.3 UPS Specials Register

Dieses Register beinhaltet spezielle Werte der USV und liegt auf I/O Port 4158h – wird von der USV IF Option nicht verwendet.

UPS Specials Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

ID	Name / Funk-tion	Default	Mode	Bemerkung

Tabelle 77: UPS Specials Register

5.7 Panel Switch Register

Dieses Register dient zum Umschalten der **Panel Service** Register des MTCX (siehe Seite 50) und liegt auf I/O Port 415Ch.

Der MTCX stellt einige Daten der Panels – z.B. Temperatur, Tastenmatrix - über Register zur Verfügung. Dabei teilen sich alle Panels einen gemeinsamen Registerbereich. Über das **Panel Switch** Register kann eingestellt werden, von welchem Panel die Daten in diesem Registerbereich bereitgestellt werden. Die Daten stehen sofort nach dem Beschreiben des Panel Switch Register zur Verfügung.

An der Monitor / Panel Schnittstelle angeschlossene Automation Panels werden ab Panelnummer 0 adressiert, am optionalen Display Link angeschlossene Automation Panels ab Panelnummer 8. Die Displayeinheit eines PPC900 wird über Panelnummer 15 adressiert.

Panel Switch Register																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	A				x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

ID	Name / Funktion	Default	Mode	Bemerkung
A	Panel Number	0	R/W	Panelnummer: 0 = 1. Panel, 1 = 2. Panel etc.

Tabelle 78: MTCX Panel Switch Register

5.8 Panel Service Register

Diese Register beinhalten die Daten und Einstellungen des mit dem **Panel Switch** Register (siehe Seite 49) ausgewählten Panels und liegen ab I/O Port 4160h.

Die Daten für die einzelnen Panels werden vom MTCX zyklisch im System abgefragt und in die Panel Service Register abgelegt. Somit kann die PC Anwendung sofort auf die zuletzt gelesenen Daten zugreifen. Die Registerdaten sind gültig, sobald ein Panel einmal erkannt („detected“) wurde und bleiben auch erhalten, wenn das Panel nicht mehr angeschlossen („linked“) ist.

Information:

- **Bevor auf diese Register zugegriffen wird, muss über das Panel Switch Register das gewünschte Panel eingestellt werden.**
- **Wenn Sie in Ihrer PC Anwendung auf panelspezifische Register aus mehreren Threads gleichzeitig zugreifen, müssen diese Zugriffe synchronisiert werden.**

4160h+ Offset	Register	Beschreibung
0000h	Panel Version	Version des Panels
0004h	Panel TempFan	Temperatur & Lüfter Überwachung
0008h	Panel Specials	LED/Tasten Anzahl, Schlüsselschalter
000Ch	Panel Flags	Sperre, Linkstatus...
0010h	Panel Key Matrix 0	Tastenmatrix: Byte 0 bis 3
0014h	Panel Key Matrix 1	Tastenmatrix: Byte 4 bis 7
0018h	Panel Key Matrix 2	Tastenmatrix: Byte 8 bis 11
001Ch	Panel Key Matrix 3	Tastenmatrix: Byte 12 bis 15

Tabelle 79: MTCX Panel Service Register Übersicht

Die einzelnen Register sind in den folgenden Kapiteln beschrieben.

5.8.1 Panel Version Register

Dieses Register beinhaltet die SDL Firmware Version des Panels und liegt auf I/O Port 4160h.

Information:

- **Bevor auf dieses Register zugegriffen wird, muss über das Panel Switch Register das gewünschte Panel eingestellt werden.**
- **Für die Displayeinheit eines PPC900 (Panel 15) wird keine gültige Version geliefert.**
- **Für AP800 Extension Units wird die Version der Base Unit geliefert.**

Panel Version Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C																x	x	x	x	B				A							

ID	Name / Funktion	Default	Mode	Bemerkung
A	Version Minor		RO	Höherwertige Versionsnummer
B	Version Major		RO	Niederwertige Versionsnummer
C	Device ID		RO	Gerätekennung – gibt die Art des Panels an: 722Ch = AP900 SDL Receiver/Transceiver 2C6Bh = AP900 LDL Receiver/Transceiver (obsolet) E05Dh = AP800 SDL Receiver (nur auf Base Unit) E0DDh = AP800 Extension Unit 0DE0h = AP830 SDL Receiver E035h = AP9x3 SDL Receiver 725Ch = AP9x3 SDL3 Receiver 5886h = eingebautes Display eines PP300/400, PP500, PPC300, PPC700, PPC800, PPC900

Tabelle 80: MTCX Panel Version Register

5.8.2 Panel TempFan Register

Dieses Register beinhaltet die Temperatur (Umgebungstemperatur des Displays) und Lüfterdrehzahl des ausgewählten Panels und liegt auf I/O Port 4164h.

Information:

- **Bevor auf dieses Register zugegriffen wird, muss über das Panel Switch Register das gewünschte Panel eingestellt werden.**
- **Für die Displayeinheit eines PPC900 (Panel 15) und AP800 Extension Units wird keine gültige Temperatur und Lüfterdrehzahl geliefert. Ob es sich um eine AP800 Extension Unit handelt, kann über das Panel Version Register (siehe Seite 51 und Codebeispiel auf Seite 92) ermittelt werden.**

Panel TempFan Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	B												x	x	x	x	x	x	x	x	A							

ID	Name / Funktion	Default	Mode	Bemerkung
A	Temp	0	RO	Temperatur in °C: 0 bis 127
B	Fan Speed	0	RO	Lüfterumdrehungen pro Minute / 4 (wird derzeit nicht unterstützt)

Tabelle 81: MTCX Panel TempFan Register

5.8.3 Panel Specials Register

Dieses Register beinhaltet spezielle Einstellungen und Informationen eines Panels (Hintergrundbeleuchtung, Schlüsselschalterstatus und die Anzahl der Tasten und LEDs) und liegt auf I/O Port 4168h.

Information:

Bevor auf dieses Register zugegriffen wird, muss über das Panel Switch Register das gewünschte Panel eingestellt werden.

Panel Specials Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D								C							B									A							

ID	Name / Funktion	Default	Mode	Bemerkung
A	Brightness [23]		RO	Helligkeit der Hintergrundbeleuchtung: 0 bis 100 %
B	Key Switches	FFh	RO	Schlüsselschalter Zustände: 00h bis FFh Die Offsets der Schlüsselschalter müssen in der Tastenkonfiguration mit B&R Key Editor Version 2.50 oder höher parametriert worden sein.
C	Key Count		RO	Maximale Anzahl von unterstützten Matrixtasten: 0 oder 128 0 bedeutet, dass keine Tasten unterstützt werden. Ein Wert ungleich 0 bedeutet, dass bis zu x Tasten unterstützt werden, sagt jedoch nicht aus, ob das Gerät auch wirklich so viele Tasten besitzt.
D	LED Count		RO	Maximale Anzahl von unterstützten Matrix-LEDs: 0 oder 128 0 bedeutet, dass keine LEDs unterstützt werden. Ein Wert ungleich 0 bedeutet, dass bis zu x LEDs unterstützt werden, sagt jedoch nicht aus, ob das Gerät auch wirklich so viele LEDs besitzt.

Tabelle 82: MTCX Panel Specials Register

[23] Obsolet: Zum Lesen der Helligkeit muss das Display Service Kommando des MTCX (siehe Seite 28) verwendet werden.

5.8.4 Panel Flags Register

Dieses Register beinhaltet Kennungen und spezielle Einstellungen des ausgewählten Panels und liegt auf I/O Port 416Ch.

Information:

Bevor auf dieses Register zugegriffen wird, muss über das Panel Switch Register das gewünschte Panel eingestellt werden.

Panel Flags Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	I	H				x	x	x	G	F				x	x	x	x	x	x	x	x	x	x	x	E	D	C	B	A

ID	Name / Funktion	Default	Mode	Bemerkung
A	Detected	0	RO	Zeigt an, ob das Panel einmal erkannt wurde: 0 = nicht erkannt, 1 = erkannt
B	Linked	0	RO	Zeigt an, ob das Panel aktuell angeschlossen ist: 0 = nicht angeschlossen, 1 = angeschlossen
C	Locked	0	RO	Zeigt an, ob das Panel (Tastenmatrix, resistiver Touchscreen) gesperrt ist: 0 = nicht gesperrt, 1 = gesperrt
D	Scancode Lock	0	RO	Zeigt an, ob die Scancodes gesperrt sind: 0 = nicht gesperrt, 1 = gesperrt
E	Software Lock	0	RO	Zeigt an, ob das Panel per MTCX Kommando gesperrt ist: 0 = nicht gesperrt, 1 = gesperrt.
F	Equalizer Auto	0	RO	Automatischer Equalizerwert (wird vom System anhand der Kabellänge errechnet): 0 (stark) bis 15 (schwach).
G	Equalizer Support [24]	0	RO	Zeigt an, ob *Equalizer* gültig ist (d.h. unterstützt wird): 0 = ungültig, 1 = gültig
H	Equalizer User	0	R/W	Vom Anwender vorgegebener Equalizerwert: 0 (stark) bis 15 (schwach) Verwenden Sie niedrige Werte (starke Equalizereinstellung) für lange Kabel.
I	Equalizer Mode	0	R/W	Gibt an, ob *EqualizerUser* verwendet wird: 0 = nein (Automatic Mode), 1 = ja (User Mode)

Tabelle 83: MTCX Panel Flags Register

Information:

Für die Displayeinheit eines PPC900 (Panel 15) und AP800 Extension Units können keine gültigen Equalizereinstellungen gelesen bzw. gesetzt werden. Ob es sich um eine AP800 Extension Unit handelt, kann über das Panel Version Register (siehe Seite 51) ermittelt werden.

[24] Der Equalizer ist im Automation Panel eingebaut und passt das DVI Signal für unterschiedliche Kabellängen an. Der Equalizerwert wird automatisch anhand der Kabellänge ermittelt. Sie können einen anderen Equalizerwert einstellen, um die beste optische Darstellung auf dem Display zu erreichen (z.B. bei schlechter Kabelqualität oder schlechter DVI Signalqualität).

Information:

- **Eine mit dem Key Service Kommando (siehe Seite 22) geänderte Scancodesperre wird bedingt durch den internen Ablauf im MTCX erst nach kurzer Verzögerung in „Scancode Lock“ im Panel Flags Register angezeigt.**
- **Eine mit dem Key Service Kommando (siehe Seite 22) geänderte Panelsperre wird bedingt durch den internen Ablauf im MTCX erst nach kurzer Verzögerung in „Locked“ und „Software Lock“ im Panel Flags Register angezeigt.**

5.8.5 Panel Key Matrix [0..3] Register

Diese Register beinhalten die Zustände der Matrixtasten des ausgewählten Panels und liegen auf I/O Port 4170h bis 417Fh.

Es gibt 4 Register für die gesamte Matrix. Die Tastenmatrix ist bereits entprellt.

Information:

Bevor auf dieses Register zugegriffen wird, muss über das Panel Switch Register das gewünschte Panel eingestellt werden.
Die Register mit den Tastenzuständen können auch gelesen werden, wenn das Gerät gar keine Tasten unterstützt. Dieser Fall muss eventuell in der Anwendung abgefangen werden, z.B. durch vorheriges Lesen der Tastenanzahl vom Panel Specials Register des MTCX (siehe Seite 52).

Panel Key Matrix [0..3] Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A																															

ID	Name / Funktion	Default	Mode	Bemerkung
A	Matrix		RO	In der Tasten Matrix werden pro Register bis zu 32 Tastenzustände angezeigt. Eine gedrückte Taste wird mit 0 gekennzeichnet.

Tabelle 84: MTCX Panel Key Matrix [0..3] Register

6 Codierhinweise

In den nächsten Kapiteln finden Sie allgemeine Hinweise für die Anwendung der in diesem Dokument enthaltenen Codebeispiele.

6.1 Type-Punning Problem

In den nachfolgenden Codebeispielen wird beim Lesen der MTCX Register an einigen Stellen eine spezielle cast Anweisung verwendet, z.B.:

```
MTCX_BASEBOARD_TEMP_2ND_REG reg;

*(unsigned long*)&reg = MTCX_READ_PORT_ULONG(MTCX_BASEBOARD_TEMP_2ND_ADDR);
*Temp = reg.Temp1;
```

Listing 1: MTCX Register lesen mit type-punning cast

Diese cast Anweisungen führen ein sogenanntes „type-punning“ durch (siehe http://en.wikipedia.org/wiki/Type_punning).

Information:

Dieses type-punning kann abhängig vom Compiler und den verwendeten Compiler Einstellungen (Optimizerstufe) entweder zu Warnungen oder Fehlern beim Übersetzen oder sogar – da Code „wegoptimiert“ wurde – zu einem nicht funktionsfähigen Programm führen!

Hinweis: In Debug Builds und mit Microsoft Compilern tritt dieses Problem nicht auf.

Sie können dieses Problem entweder durch Ausschalten der Optimierung (beim GCC Compiler z.B. mit -fno-strict-aliasing, siehe auch http://gcc.gnu.org/onlinedocs/gcc-4.1.1/gcc/Optimize-Options.html#index-fstrict_002daliasing-542) oder Umschreiben des Codes beseitigen:

```
union t_u
{
    unsigned long l;
    MTCX_BASEBOARD_TEMP_2ND_REG reg;
} u;

u.l = MTCX_READ_PORT_ULONG(MTCX_BASEBOARD_TEMP_2ND_ADDR);
*Temp = u.reg.Temp1;
```

Listing 2: MTCX Register lesen ohne type-punning cast

Aus Kompatibilität mit älteren Implementierungsanleitungen und bereits erfolgten Portierungen wird in den Codebeispielen der cast weiterhin verwendet.

6.2 TODO: Anweisungen

In den Codebeispielen sind TODO: Anweisungen enthalten. Ersetzen Sie diese durch entsprechenden Code.

6.3 Datenformate

Die in den Codebeispielen verwendeten Standard C Datentypen müssen folgende Größen besitzen:

Datenformat	Größe in Bytes
char	1
short	2
long	4
int	>= 2

Tabelle 85: Verwendete Datenformate

6.4 READ_PORT_x, WRITE_PORT_x

Sie müssen die in den Codebeispielen enthaltenen READ_PORT_x und WRITE_PORT_x Funktionen durch die entsprechenden I/O Port Zugriffsfunktionen Ihrer Entwicklungsumgebung ersetzen.

Beispiel für Implementierung der READ_PORT_x und WRITE_PORT_x Funktionen mit **inp** und **outp** (z.B. für MS-DOS):

```
//
// I/O port access functions.
//

#define READ_PORT_UCHAR(port) inp(port)
#define WRITE_PORT_UCHAR(port, value) outp(port, value)
#define READ_PORT_USHORT(port) inpw(port)
#define WRITE_PORT_USHORT(port, value) outpw(port, value)

typedef union
{
    unsigned long ldata;
    unsigned char bdata[4];
} BYTES_AND_LONG;

// Read unsigned long value from I/O port.
unsigned long READ_PORT_ULONG(unsigned short port)
{
    BYTES_AND_LONG data;

    data.bdata[0] = inp(port++);
    data.bdata[1] = inp(port++);
    data.bdata[2] = inp(port++);
    data.bdata[3] = inp(port++);
    return data.ldata;
}

// Write unsigned long value to I/O port.
void WRITE_PORT_ULONG(unsigned short port, unsigned long value)
{
    BYTES_AND_LONG data;

    data.ldata = value;
    outp(port++, (unsigned)data.bdata[0]);
    outp(port++, (unsigned)data.bdata[1]);
    outp(port++, (unsigned)data.bdata[2]);
    outp(port++, (unsigned)data.bdata[3]);
}
```

Listing 3: Beispielimplementierung der READ_PORT_x und WRITE_PORT_x Funktionen

6.5 READ_REGISTER_x

Sie müssen die in den Codebeispielen enthaltene READ_REGISTER_x Funktion durch die entsprechende Speicherzugriffsfunktion Ihrer Entwicklungsumgebung ersetzen.

Beispiel für Implementierung der READ_REGISTER_x Funktionen für MS-DOS:

```
// Convert linear address into segment offset address
// for example 000F0000h -> F000:0000
unsigned char READ_REGISTER_UCHAR(unsigned long Register)
{
    return *(char far*)(((Register & 0xFFFF0000) << 12) + (Register & 0x0000FFFF));
}
```

Listing 4: Beispielimplementierung der READ_REGISTER_UCHAR Funktion

7 Codebeispiele

In den folgenden Kapiteln sind die Codebeispiele für die gerätespezifischen Funktionen beschrieben, z.B.: Lesen von Temperaturen und Geräteinformationen.

7.1 Hilfsfunktionen

Die folgenden Codebeispiele zeigen Hilfsfunktionen, die in den Codebeispielen verwendet werden.

7.1.1 2-Byte Wert swappen

Das folgende C Codebeispiel zeigt, wie die Bytereihenfolge eines 2-Byte Werts geändert werden kann.

Diese Funktion wird z.B. beim Lesen der Geräteinformationen eines Automation Panels benötigt, da einige Daten im Big Endian Format abgelegt sind.

```
// Swap 2 byte value.
unsigned short swapw(unsigned short w)
{
    return (w >> 8) + (w << 8);
}
```

Listing 5: swapw – 2-Byte Wert swappen

7.1.2 4-Byte Wert swappen

Das folgende C Codebeispiel zeigt, wie die Bytereihenfolge eines 4-Byte Werts geändert werden kann.

Diese Funktion wird z.B. beim Lesen der Geräteinformationen eines Automation Panels benötigt, da einige Daten im Big Endian Format abgelegt sind.

```
// Swap 4 byte value.
unsigned long swapl(unsigned long dw)
{
    return swapw((unsigned short)(dw >> 16)) + ((unsigned long)swapw((unsigned short)dw) << 16);
}
```

Listing 6: swapl – 4-Byte Wert swappen

7.2 MTCX Schnittstellenfunktionen

Die folgenden Codebeispiele zeigen, wie die MTCX Schnittstelle in C angesprochen werden kann.

7.2.1 Definitionen für MTCX Schnittstelle

Dieses Listing enthält allgemeine C Definitionen der MTCX Schnittstelle: I/O Portadressen, Strukturen der MTCX Register usw.

Diese Definitionen werden in den Codebeispielen in diesem Dokument verwendet. Sie können diese Definitionen als Datei BrMtcx.h speichern und in Ihrem C Code inkludieren.

```
// BrMtcx.h : Definitions for MTCX interface on APC910 and PPC900.
//
// Copyright (c) Bernecker + Rainer

#ifndef _BRMTCX_H_
#define _BRMTCX_H_

//
// MTCX I/O register addresses.
//

#define MTCX_BASE_ADDR                  0x4100

#define MTCX_VERSION_ADDR               (MTCX_BASE_ADDR + 0x00)
#define MTCX_CONFIG_COM_ADDR            (MTCX_BASE_ADDR + 0x04)
#define MTCX_CONFIG_SPECIALS_ADDR       (MTCX_BASE_ADDR + 0x08)
#define MTCX_CONFIG_2ND_ADDR            (MTCX_BASE_ADDR + 0x0C)

#define MTCX_CMD_STATUS_ADDR            (MTCX_BASE_ADDR + 0x10)
#define MTCX_CMD_PARAM_ADDR             (MTCX_BASE_ADDR + 0x14)
#define MTCX_CMD_DATA0_ADDR             (MTCX_BASE_ADDR + 0x18)
#define MTCX_CMD_DATA1_ADDR             (MTCX_BASE_ADDR + 0x1C)
#define MTCX_CMD_DATA2_ADDR             (MTCX_BASE_ADDR + 0x20)
#define MTCX_CMD_DATA3_ADDR             (MTCX_BASE_ADDR + 0x24)

#define MTCX_HW_INFO_ADDR               (MTCX_BASE_ADDR + 0x28)
#define MTCX_HW_INFO2_ADDR              (MTCX_BASE_ADDR + 0x2C)

#define MTCX_BASEBOARD_SUPPORT_ADDR     (MTCX_BASE_ADDR + 0x30)
#define MTCX_BASEBOARD_KEYSTATE_ADDR    (MTCX_BASE_ADDR + 0x48)
#define MTCX_BASEBOARD_SPECIALS_ADDR    (MTCX_BASE_ADDR + 0x4C)

#define MTCX_UPS_STATUS_ADDR            (MTCX_BASE_ADDR + 0x50)
#define MTCX_UPS_VALUES_ADDR            (MTCX_BASE_ADDR + 0x54)
#define MTCX_UPS_SPECIALS_ADDR          (MTCX_BASE_ADDR + 0x58)

#define MTCX_PANEL_SWITCH_ADDR          (MTCX_BASE_ADDR + 0x5C)
#define MTCX_PANEL_VERSION_ADDR         (MTCX_BASE_ADDR + 0x60)
#define MTCX_PANEL_TEMPFAN_ADDR         (MTCX_BASE_ADDR + 0x64)
#define MTCX_PANEL_SPECIALS_ADDR        (MTCX_BASE_ADDR + 0x68)
#define MTCX_PANEL_FLAGS_ADDR           (MTCX_BASE_ADDR + 0x6C)
#define MTCX_PANEL_KEY_MATRIX0_ADDR     (MTCX_BASE_ADDR + 0x70)
#define MTCX_PANEL_KEY_MATRIX1_ADDR     (MTCX_BASE_ADDR + 0x74)
#define MTCX_PANEL_KEY_MATRIX2_ADDR     (MTCX_BASE_ADDR + 0x78)
#define MTCX_PANEL_KEY_MATRIX3_ADDR     (MTCX_BASE_ADDR + 0x7C)

//
// MTCX command codes.
//
#define MTCX_CMD_VERSION_INFO       0x10
#define MTCX_CMD_DEVICE_INFO        0x12
#define MTCX_CMD_KEY_LAYER          0x18
#define MTCX_CMD_KEY_SERVICE        0x19
#define MTCX_CMD_FLASH_SERVICE      0x20
#define MTCX_CMD_STATISTICS_INFO    0x30
#define MTCX_CMD_LED_SERVICE0       0x40
```

```
#define MTCX_CMD_LED_SERVICE1       0x41
#define MTCX_CMD_LED_SERVICE2       0x42
#define MTCX_CMD_LED_SERVICE3       0x43
#define MTCX_CMD_DISPLAY_SERVICE    0x48
#define MTCX_CMD_WDOG_SERVICE       0x50
#define MTCX_CMD_MODULE_INFO        0x61
#define MTCX_CMD_MODULE_TEMPERATURE 0x64
#define MTCX_CMD_MODULE_STATISTICS  0x65
#define MTCX_CMD_MODULE_CORR        0x66
#define MTCX_CMD_MODULE_VOLTAGE     0x68
#define MTCX_CMD_MODULE_HEADER      0x6A
#define MTCX_CMD_MODULE_FAN         0x6B
#define MTCX_CMD_UPS_SERVICE        0x90

//
// MTCX command (target) device types.
//
#define MTCX_DEV_COMMON     0x00 // obsolet
#define MTCX_DEV_BASEBOARD  0x01
#define MTCX_DEV_SCANCODE   0x02 // obsolet
#define MTCX_DEV_PANEL      0x03
#define MTCX_DEV_PANELSET   0x04
#define MTCX_DEV_UPS        0x08

//
// MTCX command (baseboard) device numbers.
//
#define MTCX_DEVNUM_SCAN     0x02   // Scan codes (key configuration)
#define MTCX_DEVNUM_SET      0x03   // Settings

//
// MTCX command (module) device numbers
// for r/w of temperatures, fans, factory info, statistics
//
#define MTCX_DEVNUM_APC910_PPC900_SYS     0 // system unit
#define MTCX_DEVNUM_PPC900_DISPLAY        1 // display unit
#define MTCX_DEVNUM_APC910_PPC900_BUS     2 // bus unit
#define MTCX_DEVNUM_APC910_LINK           3 // Display Link
#define MTCX_DEVNUM_APC910_PPC900_IF1     4 // IF option 1
#define MTCX_DEVNUM_APC910_PPC900_IF2     5 // IF option 2
#define MTCX_DEVNUM_APC910_PPC900_MEM1    6 // DRAM module 1
#define MTCX_DEVNUM_APC910_PPC900_MEM2    7 // DRAM module 2
#define MTCX_DEVNUM_APC910_PPC900_FAN     8 // fan kit
#define MTCX_DEVNUM_APC910_PPC900_DRIVE1  9 // slide-in slot 1
#define MTCX_DEVNUM_APC910_DRIVE2        10 // slide-in slot 2
#define MTCX_DEVNUM_APC910_PPC900_CPU    11 // CPU board
#define MTCX_DEVNUM_PPC900_FAN2          12 // fan kit 2 (bus fans)
// next defines are for compatibility:
#define MTCX_DEVNUM_APC910_SYS     0 // system unit
#define MTCX_DEVNUM_APC910_BUS     2 // bus unit
#define MTCX_DEVNUM_APC910_LINK    3 // Display Link
#define MTCX_DEVNUM_APC910_IF1     4 // IF option 1
#define MTCX_DEVNUM_APC910_IF2     5 // IF option 2
#define MTCX_DEVNUM_APC910_MEM1    6 // DRAM module 1
#define MTCX_DEVNUM_APC910_MEM2    7 // DRAM module 2
#define MTCX_DEVNUM_APC910_FAN     8 // fan kit
#define MTCX_DEVNUM_APC910_DRIVE1  9 // slide-in slot 1
#define MTCX_DEVNUM_APC910_DRIVE2 10 // slide-in slot 2
#define MTCX_DEVNUM_APC910_CPU    11 // CPU board

//
// MTCX command directions.
//
#define MTCX_DIR_READ    1   // data read command
#define MTCX_DIR_WRITE   0   // data write command

//
// MTCX command error codes (returned at MTCX_FAIL_COMMAND).
//
#define MTCX_ERR_NONE               0x00
#define MTCX_ERR_IRQ_COLLISION      0x80
```

```
#define MTCX_ERR_UNKNOWN_COMMAND    0x81
#define MTCX_ERR_TIMEOUT            0x82
#define MTCX_ERR_BAD_PARAMETER      0x83
#define MTCX_ERR_DEVICE_BUSY        0x84
#define MTCX_ERR_DEVICE_FAILURE     0x85
#define MTCX_ERR_DATA_FAILURE       0x86

//
// MTCX version IDs.
//
#define MTXC_VER_ID_SDLR 0x722c      // AP900 SDL Receiver/Transceiver
#define MTCX_VER_ID_LDLR 0x2c6b      // AP900 LDL Receiver/Transceiver (obsolete)
#define MTCX_VER_ID_AP8H 0xe05d      // AP800 SDL Receiver
#define MTCX_VER_ID_AP8X 0xe0dd      // AP800 Extension Unit
#define MTCX_VER_ID_DISP 0x5886      // built-in display
#define MTCX_VER_ID_AP83 0x0de0      // AP830 SDL Receiver
#define MTCX_VER_ID_AP9X 0xe035      // AP9X3 SDL Receiver
#define MTCX_VER_ID_AP9X_SDL3 0x725c // AP9X3 SDL3 Receiver

//
// MTCX UPS status flags.
//
#define MTCX_UPS_STATUS_POWER_OK      0x0001
#define MTCX_UPS_STATUS_BAT_SETTINGS  0x0002
#define MTCX_UPS_STATUS_ON_BAT        0x0004
#define MTCX_UPS_STATUS_LOW_BAT       0x0008
#define MTCX_UPS_STATUS_BAT_POLARITY  0x0010
#define MTCX_UPS_STATUS_BAT_FAILURE   0x0020
#define MTCX_UPS_STATUS_BAT_LIFETIME  0x0040
#define MTCX_UPS_STATUS_SHUTDOWN      0x0080
#define MTCX_UPS_STATUS_OVER_CURRENT  0x0100
#define MTCX_UPS_STATUS_FACT_SETTINGS 0x0200
#define MTCX_UPS_STATUS_USER_SETTINGS 0x0400
#define MTCX_UPS_STATUS_BAT_TEMP      0x8000

//
// MTCX register definitions.
//

typedef struct
{
    unsigned long Minor     :8;
    unsigned long Major     :4;
    unsigned long Reserve   :20;
} MTCX_VERSION_REG;

typedef struct
{
    unsigned long DisplayNo         :4; // display number, see MTCX_DISPLAY_* defines
    unsigned long DisplayFound      :1; // 0 = no display, 1 = display found
    unsigned long Reserve1          :3;
    unsigned long RunLedState       :1; // Run LED: 0 = off, 1 = on
    unsigned long Reserve2          :23;
} MTCX_CONFIG_2ND_REG;

typedef struct
{
    unsigned long Switches          :8; // Mode/Node switches
    unsigned long Reserve1          :4;
    unsigned long PanelCount        :1; // 0 = 16 panels, 1 = 1 panel
    unsigned long Reserve2          :19;
} MTCX_CONFIG_SPECIALS_REG;

typedef struct
{
    unsigned long Addr      :14;    // Address/Size depending on command; Flash address / 16
    unsigned long Dir       :1;     // 1 = Read, 0 = Write (see MTCX_DIR_x constants)
    unsigned long Reserve   :1;
    unsigned long DevNum    :4;     // see MTCX_DEVNUM_x constants
    unsigned long Target    :4;     // see MTCX_DEV_x constants
    unsigned long Command   :8;     // see MTCX_CMD_x constants
```

```
} MTCX_CMD_PARAM_REG;

typedef struct
{
    unsigned long Error     :8;     // Valid if Status = 011b (see MTCX_ERR_x constants)
    unsigned long Status    :3;     // Command service status:
                                    // 000b = free
                                    // 001b = request finished
                                    // 010b = reserved
                                    // 011b = request failed
                                    // 100b-111b = reserved
    unsigned long Lock      :1;     // 0 = command service free, 1 = locked
    unsigned long Reserve   :20;
} MTCX_CMD_STATUS_REG;

// This structure contains the status byte of the MTCX_CMD_STATUS_REG.
typedef struct
{
    unsigned char Status    :3;
    unsigned char Lock      :1;
    unsigned char Reserve   :4;
} MTCX_CMD_STATUS_REG_BYTE1;

//
// MTCX device types.
//
#define MTCX_DEVICE_APC620    0x01
#define MTCX_DEVICE_APC620E   0x02
#define MTCX_DEVICE_PPC700    0x03
#define MTCX_DEVICE_APC810    0x04
#define MTCX_DEVICE_APC820    0x05
#define MTCX_DEVICE_PPC800    0x06
#define MTCX_DEVICE_PPC300    0x07
#define MTCX_DEVICE_PP300_400 0x08
#define MTCX_DEVICE_PP500     0x0A
#define MTCX_DEVICE_APC510    0x0C
#define MTCX_DEVICE_APC511    0x0D
#define MTCX_DEVICE_APC910    0x0E
#define MTCX_DEVICE_PPC900    0x0F

typedef struct
{
    unsigned long DeviceType     : 8;   // 00h and FFh: not supported
    unsigned long MtcxBootArea   : 1;   // MTCX booted from: 0 = low area, 1 = high area
    unsigned long BiosBootArea   : 1;   // BIOS booted from: 0 = backup area, 1 = normal (update)
area
    unsigned long FpgaBootArea   : 1;   // FPGA booted from: 0 = low area, 1 = high area
                                        // PP500/APC51x: I/O board, APC910/PPC900: Display Link
    unsigned long Reserve        : 21;
 } MTCX_HW_INFO_REG;

typedef struct
{
    unsigned long Reserve1      :9;
    unsigned long NmiSkipReset  :7;     // NMI reset skip:
                                        // bit 0 = Watchdog
                                        // bit 2 = Reset switch
                                        // bit 3 = Software reset (command)
    unsigned long SmcStatus     :6;     // SMC status:
                                        // bit 0 = Watchdog
                                        // bit 1 = Power fail
                                        // bit 2 = Reset switch
                                        // bit 3 = Software reset (command)
                                        // bit 4 = Power button
    unsigned long NmiIrqActive  :1;     // 1 = NMI IRQ active
    unsigned long NmiStatusInval:1;     // 1 = NMI status invalid
    unsigned long WatchdogToggle:1;     // set 0 to toggle watchdog
    unsigned long Reserve2      :7;
} MTCX_BASEBOARD_SUPPORT_REG;

typedef struct
```

```
{
    unsigned long Layer     :4;     // 0 to 3
    unsigned long Mode      :4;
    unsigned long Checksum  :8;     // Checksum of key configuration
    unsigned long Status    :1;     // 0 = Fail, 1 = Okay
    unsigned long Reserve   :15;
} MTCX_BASEBOARD_KEYSTATE_REG;

typedef struct
{
    unsigned long Reserve1      :8;
    unsigned long BatState      :2; // CMOS battery state: 00b = good, 11b = bad
    unsigned long PowerButton   :1; // State of power button: 0 = pressed, 1 = not pressed
    unsigned long ResetButton   :1; // State of reset button: 0 = pressed, 1 = not pressed
    unsigned long Reserve2      :4;
    unsigned long ResetCounter  :8;
    unsigned long Reserve3      :8;
} MTCX_BASEBOARD_SPECIALS_REG;

typedef struct
{
    unsigned long Detected   :1;
    unsigned long Linked     :1;
    unsigned long Reserve1   :6;
    unsigned long Flags      :16; // see MTCX_UPS_STATUS_x constants
    long          BatTemp    :8;  // 0 to 127 °C
} MTCX_UPS_STATUS_REG;

typedef struct
{
    unsigned char Detected   :1;
    unsigned char Linked     :1;
    unsigned char Reserve1   :6;
} MTCX_UPS_STATUS_REG_BYTE0;

typedef struct
{
    unsigned short PowerOk      :1;
    unsigned short BatSettings  :1;
    unsigned short OnBat        :1;
    unsigned short LowBat       :1;
    unsigned short BatPolarity  :1;
    unsigned short BatFailure   :1;
    unsigned short BatLifetime  :1;
    unsigned short Shutdown     :1;
    unsigned short Reserve2     :7;
    unsigned short BatTempAlarm :1;
} MTCX_UPS_STATUS_FLAGS;

typedef struct
{
    long          BatCurrent  :16; // pos. value = charge current in mA
                                   // neg. value = discharge current in mA
    unsigned long BatVoltage  :16; // mV
} MTCX_UPS_VALUES_REG2; // Values register for IF UPS on APC910 and PPC900

typedef struct
{
    unsigned long Minor     :8;
    unsigned long Major     :4;
    unsigned long Reserve   :4;
    unsigned long Id        :16;    // see MTCX_VER_ID_x
} MTCX_PANEL_VERSION_REG;

typedef struct
{
    unsigned long Reserve       :8;
    unsigned long KeySwitches   :8; // 00h to ffh
    unsigned long KeyCount      :8; // 0 or 128
    unsigned long LedCount      :8; // 0 or 128
} MTCX_PANEL_SPECIALS_REG;
```

```
typedef struct
{
    unsigned long Detected          :1;
    unsigned long Linked            :1;
    unsigned long Locked            :1;
    unsigned long ScanCodeLock      :1;
    unsigned long SoftwareLock      :1;
    unsigned long Reserve1          :11;
    unsigned long EqualizerAuto     :4;
    unsigned long EqualizerSupport  :1;
    unsigned long Reserve2          :3;
    unsigned long EqualizerUser     :4;
    unsigned long EqualizerMode     :1;
    unsigned long Reserve3          :3;
} MTCX_PANEL_FLAGS_REG;

typedef struct
{
    unsigned char Detected          :1;
    unsigned char Linked            :1;
    unsigned char Locked            :1;
    unsigned char ScanCodeLock      :1;
    unsigned char SoftwareLock      :1;
    unsigned char Reserve           :3;
} MTCX_PANEL_FLAGS_REG_BYTE0; // byte 0 of register

typedef struct
{
    long          Temp      :8;     // 0 to 127
    unsigned long Reserve1  :8;
    unsigned long FanSpeed  :12;    // RPM / 4
    unsigned long Reserve2  :4;
} MTCX_PANEL_TEMPFAN_REG;

//
// MTCX command data definitions.
//

#define MTCX_CMD_DATA_SIZE    16        // Maximum command data size

#pragma pack(1)

typedef struct
{
    unsigned char   Brightness; // brightness (backlight) in percent: 0 to 100,
                                // 255 = not supported
    unsigned char   Reserve1;
    unsigned short  Reserve2;
} MTCX_DISPLAY_SERVICE_DATA;

// Version info command returns same data as version register.
#define MTCX_VERSION_INFO_DATA MTCX_VERSION_REG

typedef struct
{
    unsigned long  DeviceId;
    unsigned short CompatId;
    unsigned long  VendorId;
    char           HwRevision[5]; // incl. 0 byte
    char           Reserve;
} MTCX_MANUFACTURER_INFO_DATA;

typedef struct
{
    unsigned long  DeviceId;
    unsigned short CompatId;
    char           Reserve[2];
} MTCX_PARENT_INFO_DATA;

typedef struct
```

```
{
    char SerialNumber[12]; // incl. 0 byte
    char Reserve[4];
} MTCX_SERIAL_NUMBER_DATA;

typedef struct
{
    unsigned long PowerOnHours; // 1/4 h
    unsigned long PowerOnCycles;
} MTCX_MODULE_STATISTICS_DATA;

typedef struct
{
    unsigned long BacklightOnHours; // 1/4 h
    unsigned long BacklightOnCycles;
} MTCX_DISPLAY_STATISTICS_DATA;

typedef struct
{
    unsigned long FanOnHours; // 1/4 h
    unsigned long FanOnCycles;
} MTCX_FAN_STATISTICS_DATA;

typedef struct
{
    unsigned long OnBatHours; // 1/4 h
    unsigned long OnBatCycles;
} MTCX_UPS_STATISTICS_DATA;

typedef struct
{
    unsigned short BacklightOnCycles;
    unsigned short BacklightOnHours;
} MTCX_PANEL_STATISTICS_DATA;

typedef struct
{
    unsigned short MinTime;  // 0 to 65535 ms
    unsigned short MaxTime;  // 0 to 65535 ms
} MTCX_WATCHDOG_CONFIG_DATA;

typedef struct
{
    unsigned long Reserve;
    unsigned long User;
} MTCX_ID_SERVICE_DATA;

typedef struct
{
    unsigned short LockTime; // 0 to 65535 ms
    unsigned char  Reserve[2];
} MTCX_PANEL_LOCK_TIME_DATA;

typedef struct
{
    unsigned short LockBits; // Bit x = 1: panel x locked
    unsigned short MaskBits; // Bit x = 1: use LockBits bit x
} MTCX_PANEL_LOCK_DATA;

typedef struct
{
    unsigned char Layer; // 0 to 3
    unsigned char Reserve[3];
} MTCX_KEY_LAYER_DATA;

typedef struct
{
    unsigned short LockBits; // Bit x = 1: scan codes of panel x locked
    unsigned short Reserve;
} MTCX_SCANCODE_LOCK_DATA;
```

```
typedef struct
{
    unsigned short Delay; // 10 to 1200 s
    unsigned short Reserve;
} MTCX_UPS_SHUTDOWN_DATA;

typedef struct
{
    unsigned long Status  :8;
    unsigned long Reserve :8;
    unsigned long Flags   :6;
    long Temp             :10; // Temperature value in 0.25 degrees Celsius
} MTCX_MODULE_TEMPERATURE_DATA;

typedef struct
{
    struct
    {
        unsigned short Value :15; // Battery voltage in 10 mV
        unsigned short State :1;  // Battery status: 0 = good, 1 = bad
    } Vbat;
    struct
    {
        unsigned short Value :15; // Input voltage in 100 mV
        unsigned short State :1;
    } Vin;
    unsigned short Wcpu; // System power consumption in 10 mW
    unsigned short Reserve;
} MTCX_MODULE_VOLTAGE_DATA;

typedef struct
{
    unsigned char Status;
    unsigned char Reserve;
    unsigned short FanSpeed; // Fan speed in rpm
} MTCX_MODULE_FAN_DATA;

#pragma pack()

//
// Maximum values.
//
#define MTCX_MAX_PANELS       16        // Maximum number of panels
#define MTCX_MAX_LAYERS       4         // Maximum number of key and LED layers (per panel)
#define MTCX_MAX_KEYS         128       // Maximum number of keys per layer
#define MTCX_MAX_LEDS         128       // Maximum number of LEDs per layer

//
// Key layer modes.
//
#define MTCX_LAYER_SHIFT   0x00         // Layer shift mode
#define MTCX_LAYER_TOGGLE  0x01         // Layer toggle mode
#define MTCX_LAYER_ONESHOT 0x02         // Layer one-shot mode

//
// LED states.
//
#define MTCX_LED_OFF        0           // LED is off
#define MTCX_LED_BLINK_SLOW 1           // LED is slow blinking
#define MTCX_LED_BLINK_FAST 2           // LED is fast blinking
#define MTCX_LED_ON         3           // LED is on

#endif
```

Listing 7: Definitionen für MTCX Schnittstelle

7.2.2 Fehlercodes der MTCX Schnittstellenfunktionen

Dieses Listing enthält die Fehlercodes der MTCX Schnittstellenfunktionen.
Diese Fehlercodes werden in den Codebeispielen in diesem Dokument verwendet.

```
//
// MTCX function return values.
// TODO: These return values are used in the code examples.
//       You can replace them with your own values.
//

#define MTCX_OKAY           0
#define MTCX_WORKING        1          // MTCX is working
#define MTCX_FAIL_PARAM     -1         // Invalid function parameter
#define MTCX_FAIL_LOCKED    -2         // Command interface is locked
#define MTCX_FAIL_COMMAND   -3         // Command failed
#define MTCX_FAIL_TIMEOUT   -4         // Timeout detected
#define MTCX_FAIL_DETECT    -5         // Panel not detected
#define MTCX_FAIL_LINK      -6         // Panel not linked
#define MTCX_FAIL_DATA      -7         // Invalid data
#define MTCX_FAIL_NOSUPPORT -8         // Not supported
```

Listing 8: Fehlercodes der MTCX Schnittstellenfunktionen

7.2.3 Maximale Panelanzahl lesen

Durch Lesen des **Config Specials** Register des MTCX (siehe Seite 13) kann ermittelt werden, wie viele Panels vom MTCX unterstützt werden.

Information:

Auf APC910 und PPC900 werden vom MTCX immer max. 16 Panels unterstützt - unabhängig davon wie viele Panels an das Gerät tatsächlich angeschlossen werden können: z.B. auf einem APC910 ohne Display Link sind theoretisch max. 8 Automation Panel möglich.

```
// Return maximum number of supported panels.
int MtcxGetMaxPanelCount()
{
    MTCX_CONFIG_SPECIALS_REG reg;

    *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_CONFIG_SPECIALS_ADDR);
    if (reg.PanelCount == 1) // 1 panel
        // For example PP300/400 supports only panel number 0.
        return 1;

    return MTCX_MAX_PANELS;
}
```

Listing 9: MtcxGetMaxPanelCount – maximale Panelanzahl lessen

Die verwendeten Definitionen finden Sie auf Seite 59.

7.2.4 Panel umschalten

Durch Beschreiben des **Panel Switch** Register des MTCX (siehe Seite 49) kann das Panel ausgewählt werden, auf dessen Daten und Einstellungen über die **Panel Service** Register des MTCX anschließend zugegriffen werden kann.

Hinweis: In den Codebeispielen in diesem Dokument wird in jeder panelspezifischen Funktion das Panel umgeschaltet. Abhängig von Ihrer Anwendung können Sie das eventuell auch aus den Funktionen entfernen und z.B. beim Start der Anwendung durchführen.

```
// Switch to panel.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//
// Return MTCX_OKAY at success and MTCX_FAIL_x at failure.
int MtcxSwitchToPanel(unsigned int PanelNumber)
{
    // Check how many panels are supported.
    // TODO: you can remove this check if the function is not used
    // on a device that supports only one panel
    if (MtcxGetMaxPanelCount() == 1)
    {
        if (PanelNumber != 0)
            return MTCX_FAIL_PARAM;
    }
    // TODO: you can remove this branch if the function is not used
    // on a device that supports more than one panel
    else // 16 panels
    {
        if (PanelNumber >= MTCX_MAX_PANELS)
            return MTCX_FAIL_PARAM;
        WRITE_PORT_UCHAR(MTCX_PANEL_SWITCH_ADDR + 3, (unsigned char)PanelNumber); // write byte 3
only
    }
    return MTCX_OKAY;
}
```

Listing 10: MtcxSwitchToPanel – Panel umschalten

Die verwendeten Definitionen finden Sie auf Seite 59.

Durch das zusätzliche Auswerten der „linked“ Kennung im **Panel Flags** Register des MTCX (siehe Seite 53) kann eine Funktion erstellt werden, mit der auf ein angeschlossenes Panel umgeschaltet wird. Aus Performancegründen wird in **MtcxSwitchToLinkedPanel** ebenfalls nur das niederwertigste Byte des **Panel Flags** Register angesprochen.

```
// Switch to linked panel.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//
// Return MTCX_OKAY at success and MTCX_FAIL_x at failure.
int MtcxSwitchToLinkedPanel(unsigned int PanelNumber)
{
    int retvalue = MtcxSwitchToPanel(PanelNumber);
    if (retvalue == MTCX_OKAY)
    {
        MTCX_PANEL_FLAGS_REG_BYTE0 reg;

        *(unsigned char*)&reg = READ_PORT_UCHAR(MTCX_PANEL_FLAGS_ADDR); // read byte 0 only
        if (! reg.Detected) // never detected
            return MTCX_FAIL_DETECT;
        if (! reg.Linked) // once detected, but not linked now?
            return MTCX_FAIL_LINK;
    }
    return retvalue;
}
```

Listing 11: MtcxSwitchToLinkedPanel – auf angeschlossenes Panel umschalten

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxGetMaxPanelCount** finden Sie auf Seite 67

7.2.5 Basisfunktionen für MTCX Kommandoschnittstelle

Diese Funktionen sind für die Bearbeitung von MTCX Kommandos notwendig (Basisfunktionen) und werden von **MtcxReadCommand** und **MtcxWriteCommand** aufgerufen.

```
// Lock the command interface.
// Return 0 if interface is already locked and
// 1 if interface is locked successfully now.
int MtcxLockCommand()
{
    unsigned long reg = READ_PORT_ULONG(MTCX_CMD_STATUS_ADDR);

    if (((MTCX_CMD_STATUS_REG*)&reg)->Lock)
        return 0;

    ((MTCX_CMD_STATUS_REG*)&reg)->Lock = 1;
    ((MTCX_CMD_STATUS_REG*)&reg)->Error = 0;
    ((MTCX_CMD_STATUS_REG*)&reg)->Status = 0;
    WRITE_PORT_ULONG(MTCX_CMD_STATUS_ADDR, reg);
    return 1;
}

// Unlock the command interface.
void MtcxUnlockCommand()
{
    unsigned long reg = READ_PORT_ULONG(MTCX_CMD_STATUS_ADDR);
    ((MTCX_CMD_STATUS_REG*)&reg)->Error = 0;
    ((MTCX_CMD_STATUS_REG*)&reg)->Status = 0;
    ((MTCX_CMD_STATUS_REG*)&reg)->Lock = 0;
    WRITE_PORT_ULONG(MTCX_CMD_STATUS_ADDR, reg);
}

// Set (request) a command.
//
// Parameters
//   Dir
//     [in] Specifies the direction: 1 = read, 0 = write.
//   Command
//     [in] Specifies the command code (see MTCX_CMD_x constants).
//   Addr
//     [in] Specifies the address (depends on command).
//   DevNum
//     [in] Specifies the device number (see MTCX_DEVNUM_x constants).
//   Target
//     [in] Specifies the target device (see MTCX_DEV_x constants).
//   Data
//     [in/out] Points to the buffer that contains/receives the
//     command data. Can be NULL if no data is required.
//   Size
//     [in] Specifies the data size in bytes: 0 to 16.
//
// Return MTCX_OKAY at success and MTCX_FAIL_x at failure.
int MtcxSetCommand(unsigned int Dir,
                   unsigned int Command,
                   unsigned int Addr,
                   unsigned int DevNum,
                   unsigned int Target,
                   void *Data,
                   unsigned int Size)
{
    MTCX_CMD_PARAM_REG param;

    // Lock command interface.
    if (! MtcxLockCommand())
        return MTCX_FAIL_LOCKED;

    // Write command data.
    if (Data && Size)
    {
```

```
        int Port = MTCX_CMD_DATA0_ADDR;
        unsigned char *p = (unsigned char*)Data;

        while (Size-- > 0)
            WRITE_PORT_UCHAR(Port++, *p++);
    }

    // Write command parameters.
    param.Dir = Dir;
    param.Command = Command;
    param.Addr = Addr;
    param.DevNum = DevNum;
    param.Target = Target;

    WRITE_PORT_ULONG(MTCX_CMD_PARAM_ADDR, *(unsigned long*)&param);
    return MTCX_OKAY;
}

// Read the command status and error code.
//
// Parameters
//   ErrorCode
//     [out] Points to a variable that receives the
//     command error when the function returns with
//     MTCX_FAIL_COMMAND (see MTCX_ERR_x constants).
//
// Return MTCX_OKAY at success, MTCX_WORKING if the
// command interface is busy and MTCX_FAIL_x at failure.
int MtcxGetCommandStatus(unsigned char *ErrorCode)
{
    int retvalue;
    unsigned char regbyte;

    regbyte = READ_PORT_UCHAR(MTCX_CMD_STATUS_ADDR + 1); // read byte 1 only
    retvalue = MTCX_WORKING;
    switch (((MTCX_CMD_STATUS_REG_BYTE1*)&regbyte)->Status)
    {
        case 0x01:
            retvalue = MTCX_OKAY;
            break;
        case 0x03:
            retvalue = MTCX_FAIL_COMMAND;
            break;
    }
    *ErrorCode = READ_PORT_UCHAR(MTCX_CMD_STATUS_ADDR); // read byte 0 only
    return retvalue;
}

// Wait for command response.
//
// Parameters
//   ErrorCode
//     [out] Points to a variable that receives the
//     command error when the function returns with
//     MTCX_FAIL_COMMAND (see MTCX_ERR_x constants).
//
// Return MTCX_OKAY at success and MTCX_FAIL_x at failure.
int MtcxWaitForCommandResponse(unsigned char *ErrorCode)
{
    int retvalue;

    while ((retvalue = MtcxGetCommandStatus(ErrorCode)) == MTCX_WORKING)
    {
        // TODO: add timeout handling and terminate loop
        //       after timeout with retvalue = MTCX_FAIL_TIMEOUT
        ;
    }
    return retvalue;
}
```

Listing 12: Basisfunktionen für MTCX Kommandoschnittstelle

Die verwendeten Definitionen finden Sie auf Seite 59.

7.2.6 Daten lesen mit MTCX Kommando

Mit dieser Funktion wird ein MTCX Kommando zum Lesen von Daten abgesetzt. Diese Funktion wird in weiteren Codebeispielen aufgerufen.

```
// Read data via MTCX command.
//
// Parameters
//   Command
//     [in] Specifies the command code (see MTCX_CMD_x constants).
//   Addr
//     [in] Specifies the address (depends on command).
//   DevNum
//     [in] Specifies the device number (see MTCX_DEVNUM_x constants).
//   Target
//     [in] Specifies the target device (see MTCX_DEV_x constants).
//   Data
//     [out] Points to a buffer that receives the
//     command data. Can be NULL if no data is required.
//   Size
//     [in] Specifies the data size in bytes: 0 to 16.
//   ErrorCode
//     [out] Points to a variable that receives the
//     command error when the function returns with
//     MTCX_FAIL_COMMAND (see MTCX_ERR_x constants).
//
// Return MTCX_OKAY at success and MTCX_FAIL_x at failure.
int MtcxReadCommand(unsigned int Command,
                    unsigned int Addr,
                    unsigned int DevNum,
                    unsigned int Target,
                    void *Data,
                    unsigned int Size,
                    unsigned char *ErrorCode)
{
    int retvalue;

    if (Size > MTCX_CMD_DATA_SIZE)
        return MTCX_FAIL_PARAM;

    // TODO: acquire command synchronization object here
    //       (if multiple commands can be requested parallel)

    retvalue = MtcxSetCommand(MTCX_DIR_READ, Command, Addr, DevNum, Target, 0, 0);
    if (retvalue == MTCX_OKAY)
    {
        retvalue = MtcxWaitForCommandResponse(ErrorCode);
        if (retvalue == MTCX_OKAY)
        {
            int Port = MTCX_CMD_DATA0_ADDR;
            unsigned char *p = (unsigned char *)Data;

            while (Size-- > 0)
                *p++ = READ_PORT_UCHAR(Port++);
        }
        MtcxUnlockCommand();
    }

    // TODO: release command synchronization object here

    return retvalue;
}
```

Listing 13: MtcxReadCommand – Daten lesen mit MTCX Kommando

Die verwendeten Definitionen finden Sie auf Seite 59.

7.2.7 Daten schreiben mit MTCX Kommando

Mit dieser Funktion wird ein MTCX Kommando zum Schreiben von Daten abgesetzt. Diese Funktion wird in weiteren Codebeispielen aufgerufen.

```
// Write data via MTCX command.
//
// Parameters
//   Command
//     [in] Specifies the command code (see MTCX_CMD_x constants).
//   Addr
//     [in] Specifies the address (depends on command).
//   DevNum
//     [in] Specifies the device number (see MTCX_DEVNUM_x constants).
//   Target
//     [in] Specifies the target device (see MTCX_DEV_x constants).
//   Data
//     [in] Points to a buffer that contains the
//     command data. Can be NULL if no data is required.
//   Size
//     [in] Specifies the data size in bytes: 0 to 16.
//   ErrorCode
//     [out] Points to a variable that receives the
//     command error when the function returns with
//     MTCX_FAIL_COMMAND (see MTCX_ERR_x constants).
//
// Return MTCX_OKAY at success and MTCX_FAIL_x at failure.
int MtcxWriteCommand(unsigned int Command,
                     unsigned int Addr,
                     unsigned int DevNum,
                     unsigned int Target,
                     void *Data,
                     unsigned int Size,
                     unsigned char *ErrorCode)
{
    int retvalue;

    if (Size > MTCX_CMD_DATA_SIZE)
        return MTCX_FAIL_PARAM;

    // TODO: acquire command synchronization object here
    //       (if multiple commands can be requested parallel)

    retvalue = MtcxSetCommand(MTCX_DIR_WRITE, Command, Addr, DevNum, Target, Data, Size);
    if (retvalue == MTCX_OKAY)
    {
        retvalue = MtcxWaitForCommandResponse(ErrorCode);
        MtcxUnlockCommand();
    }

    // TODO: release command synchronization object here

    return retvalue;
}
```

Listing 14: MtcxWriteCommand – Daten schreiben mit MTCX Kommando

Die verwendeten Definitionen finden Sie auf Seite 59.

7.3 Panelfunktionen

Die Panelfunktionen werden vom MTCX ausgeführt.

An einem APC910 und PPC900 können je nach Modell ein oder mehrere Automation Panels angeschlossen werden. Diese Automation Panels werden über eine Panelnummer ab 0 (bei Anschluss an der Monitor / Panel Schnittstelle) bzw. 8 (bei Anschluss am optionalen Display Link = Monitor / Panel Option) angesprochen.

Ein PPC900 besitzt außerdem eine Displayeinheit, die über die Panelnummer 15 ebenfalls wie ein Panel angesprochen werden kann. Für dieses Panel werden technisch bedingt nicht alle Panelfunktionen unterstützt:

- Keine Firmwareversion
- Keine Statistikwerte
- Keine Geräteinformationen
- Keine Temperatur
- Kein Equalizer

Sie können aber folgende Funktionen für alle Panels ausführen:

- Panelflags lesen (Panel „erkannt“, Panel „verbunden“, Panel „gesperrt”, Scancodes „gesperrt“) – damit können Sie auch die Panelanzahl ermitteln
- Panelsperrzeit lesen und setzen
- Panelsperre lesen und setzen

Die folgenden Codebeispiele zeigen, wie Sie diese Funktionen ausführen können.

7.3.1 Panel „unterstützt“ prüfen

Ob ein Panel unterstützt wird, kann erkannt werden, indem versucht wird, auf das Panel umzuschalten.

Das folgende C Codebeispiel zeigt, wie geprüft werden kann, ob ein Panel „unterstützt“ wird.

```
// Check if panel is supported.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//     Note: PP300/400 supports only panel number 0.
//
// Return 1 if panel is detected and 0 if not.
int IsPanelSupported(unsigned int PanelNumber)
{
    int retvalue;
    int Supported = 0;

    // TODO: acquire synchronization object for panel data here

    retvalue = MtcxSwitchToPanel(PanelNumber);
    if (retvalue == MTCX_OKAY)
        Supported = 1;

    // TODO: release synchronization object for panel data here

    return Supported;
}
```

Listing 15: IsPanelSupported – Panel „unterstützt“ prüfen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxSwitchToPanel** finden Sie auf Seite 68.

7.3.2 Panel „erkannt“ prüfen

Ob ein Panel erkannt wurde, kann über das **Panel Flags** Register des MTCX (siehe Seite 53) geprüft werden.

Das folgende C Codebeispiel zeigt, wie geprüft werden kann, ob ein Panel erkannt wurde.

```
// Check if panel is detected.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//
// Return 1 if panel is detected and 0 if not.
int IsPanelDetected(unsigned int PanelNumber)
{
    int retvalue;
    int Detected = 0;

    // TODO: acquire synchronization object for panel data here

    retvalue = MtcxSwitchToPanel(PanelNumber);
    if (retvalue == MTCX_OKAY)
    {
        MTCX_PANEL_FLAGS_REG_BYTE0 reg;

        *(unsigned char*)&reg = READ_PORT_UCHAR(MTCX_PANEL_FLAGS_ADDR); // read byte 0 only
        Detected = reg.Detected;
    }

    // TODO: release synchronization object for panel data here

    return Detected;
}
```

Listing 16: IsPanelDetected – Panel “erkannt” prüfen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxSwitchToPanel** finden Sie auf Seite 68.

7.3.3 Panel „verbunden“ prüfen

Ob ein Panel verbunden ist (d.h. die Kommunikation mit dem Panel funktioniert), kann über das **Panel Flags** Register des MTCX (siehe Seite 53) geprüft werden.

Das folgende C Codebeispiel zeigt, wie geprüft werden kann, ob ein Panel verbunden ist.

```
// Check if panel is linked.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//
// Return 1 if panel is linked and 0 if not.
int IsPanelLinked(unsigned int PanelNumber)
{
    int retvalue;
    int Linked = 0;

    // TODO: acquire synchronization object for panel data here

    retvalue = MtcxSwitchToPanel(PanelNumber);
    if (retvalue == MTCX_OKAY)
    {
        MTCX_PANEL_FLAGS_REG_BYTE0 reg;

        *(unsigned char*)&reg = READ_PORT_UCHAR(MTCX_PANEL_FLAGS_ADDR); // read byte 0 only
        Linked = reg.Linked;
    }

    // TODO: release synchronization object for panel data here

    return Linked;
}
```

Listing 17: IsPanelLinked – Panel “verbunden” prüfen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxSwitchToPanel** finden Sie auf Seite 68.

7.3.4 Panel „gesperrt“ prüfen

Ob ein Panel gesperrt ist, kann über das **Panel Flags** Register des MTCX (siehe Seite 53) geprüft werden.

Das folgende C Codebeispiel zeigt, wie geprüft werden kann, ob ein Panel gesperrt ist.

```
// Check if panel is locked.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//
// Return 1 if panel is locked and 0 if not.
int IsPanelLocked(unsigned int PanelNumber)
{
    int retvalue;
    int Locked = 0;

    // TODO: acquire synchronization object for panel data here

    retvalue = MtcxSwitchToPanel(PanelNumber);
    if (retvalue == MTCX_OKAY)
    {
        MTCX_PANEL_FLAGS_REG_BYTE0 reg;

        *(unsigned char*)&reg = READ_PORT_UCHAR(MTCX_PANEL_FLAGS_ADDR); // read byte 0 only
        Locked = reg.Locked;
    }

    // TODO: release synchronization object for panel data here

    return Locked;
}
```

Listing 18: IsPanelLocked – Panel “gesperrt” prüfen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxSwitchToPanel** finden Sie auf Seite 68.

7.3.5 Scancodes „gesperrt“ prüfen

Ob die Scancodes eines Panels gesperrt sind, kann über das **Panel Flags** Register des MTCX (siehe Seite 53) geprüft werden.

Die Scancodes sind in der Tastenkonfiguration definiert. Eine Tastenkonfiguration kann mit dem B&R Key Editor erstellt und auf das Gerät geladen werden.

Das folgende C Codebeispiel zeigt, wie geprüft werden kann, ob die Scancodes eines Panels gesperrt sind.

```
// Check if scan codes are locked.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//
// Return 1 if scan codes are locked and 0 if not.
int AreScanCodesLocked(unsigned int PanelNumber)
{
    int retvalue;
    int ScanCodeLock = 0;

    // TODO: acquire synchronization object for panel data here

    retvalue = MtcxSwitchToPanel(PanelNumber);
    if (retvalue == MTCX_OKAY)
    {
        MTCX_PANEL_FLAGS_REG_BYTE0 reg;

        *(unsigned char*)&reg = READ_PORT_UCHAR(MTCX_PANEL_FLAGS_ADDR); // read byte 0 only
        ScanCodeLock = reg.ScanCodeLock;
    }

    // TODO: release synchronization object for panel data here

    return ScanCodeLock;
}
```

Listing 19: AreScanCodesLocked – Scancodes “gesperrt” prüfen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxSwitchToPanel** finden Sie auf Seite 68.

7.3.6 Panelsperrzeit lesen/setzen

Die Sperrzeit eines Panels kann mit dem **Key Service** Kommando des MTCX (siehe Seite 22) gelesen und gesetzt werden.

Die Panelsperrzeit gibt an, wie lange auf anderen Panels die Eingabe über Tasten und resistivem Touchscreen gesperrt ist, wenn auf einem Panel eine Eingabe erfolgt. Die Panelsperrzeit wird normalerweise in der mit dem B&R Key Editor erstellten Tastenkonfiguration eingestellt.

Das folgende C Codebeispiel zeigt, wie die Sperrzeit gelesen werden kann.

```
// Get lock time of panel.
//
// Parameters
//     [out] Points to a variable that receives the
//     lock time in ms: 0 to 65535.
//
// Return 0 at success and -1 at failure.
int GetPanelLockTime(unsigned short *LockTime)
{
    MTCX_PANEL_LOCK_TIME_DATA data;
    unsigned char ErrorCode;
    int retvalue;

    if ((retvalue = MtcxReadCommand(MTCX_CMD_KEY_SERVICE, 0, 3, MTCX_DEV_BASEBOARD,
        &data, sizeof(data), &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    *LockTime = data.LockTime;
    return 0;
}
```

Listing 20: GetPanelLockTime – Panelsperrzeit lesen

Das folgende C Codebeispiel zeigt, wie die Sperrzeit gesetzt werden kann.

Information:

Geben Sie als Sperrzeit entweder 0 (= keine Sperre) oder Werte ab 500 ms an.

Information:

Die Panelsperrzeit wird bei Neustart des Systems oder wenn die Tastenkonfiguration neu programmiert oder neu geladen wird, auf den in der Tastenkonfiguration parametrierten Wert zurückgesetzt.

```
// Set lock time of panel.
//
// Parameters
//   LockTime
//     [in] Specifies the lock time in ms: 0 ro 65535.
//
// Return 0 at success and -1 at failure.
int SetPanelLockTime(unsigned short LockTime)
{
    MTCX_PANEL_LOCK_TIME_DATA data;
    unsigned char ErrorCode;
    int retvalue;

    memset((void*)&data, 0x00, sizeof(data));
    data.LockTime = LockTime;

    // Note: the lock time is reset to the key configuration value
    //       at reboot of the device or when a new key configuration
    //       is downloaded (and the scan codes are "restarted").
    if ((retvalue = MtcxWriteCommand(MTCX_CMD_KEY_SERVICE, 0, 3, MTCX_DEV_BASEBOARD,
        &data, sizeof(data), &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    return 0;
}
```

Listing 21: SetPanelLockTime – Panelsperrzeit setzen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.
Die aufgerufene Funktion **MtcxWriteCommand** finden Sie auf Seite 74.

7.3.7 Panelsperre lesen/setzen

Die Sperre eines Panels kann mit dem **Key Service** Kommando des MTCX (siehe Seite 22) gelesen und manuell gesetzt werden.

Die Eingabe über Tasten und resistivem Touchscreen eines Panels ist normalerweise automatisch gesperrt, wenn auf einem anderen angeschlossenen Panel eine Eingabe erfolgt und eine Panelsperrzeit in der Tastenkonfiguration oder mit dem Key Service Kommando eingestellt ist. Nach Ablauf der Panelsperrzeit ist eine Eingabe wieder möglich.

Das folgende C Codebeispiel zeigt, wie die Sperre eines angeschlossenen Panels gelesen werden kann.

Locked = 1 zeigt an, dass das Panel gesperrt ist. Eingaben über Tasten oder resistivem Touchsreen des Panels werden dann nicht ausgewertet. Bei Locked = 0 ist das Panel nicht gesperrt.

Ob ein Panel gesperrt ist, kann auch schneller über das **Panel Flags** Register des MTCX (siehe Seite 53) ermittelt werden (siehe **MtcxIsPanelLocked** Codebeispiel auf Seite 79).

```
// Get lock state of panel.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   Locked
//     [out] Points to a variable that receives the
//     lock state: 1 = locked, 0 = unlocked.
//
// Return 0 at success and -1 at failure.
int GetPanelLock(unsigned int PanelNumber, int *Locked)
{
    MTCX_PANEL_LOCK_DATA data;
    unsigned char ErrorCode;
    int retvalue;

    if ((retvalue = MtcxReadCommand(MTCX_CMD_KEY_SERVICE, 0, 4, MTCX_DEV_BASEBOARD,
        &data, sizeof(data), &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    // Note: LockBits bit x represents the lock state of panel x:
    //       bit value 0 = lock OFF, 1 = lock ON.
    *Locked = (data.LockBits & (1<<PanelNumber)) ? 1 : 0;
    return 0;
}
```

Listing 22: GetPanelLock – Panelsperre lesen

Das folgende C Codebeispiel zeigt, wie die Sperre eines angeschlossenen Panels gesetzt werden kann.

Wenn Lock 1 ist, wird die Eingabe über Tasten und resistivem Touchscreen gesperrt. Es können dann die Zustände der Tasten auch nicht durch Lesen der Tastenmatrix ermittelt werden. Wenn Lock 0 ist, wird die Eingabe wieder freigegeben.

```
// Set lock state of panel.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   Lock
//     [in] Specifies the lock state: 1 = locked, 0 = unlocked.
//
// Return 0 at success and -1 at failure.
int SetPanelLock(unsigned int PanelNumber, int Lock)
{
    MTCX_PANEL_LOCK_DATA data;
    unsigned char ErrorCode;
    int retvalue;

    // Note: LockBits bit x represents the lock state of panel x:
    //       bit value 0 = lock OFF, 1 = lock ON.
    //       A bit is evaluated by the MTCX only when the corresponding
    //       bit in MaskBits is set 1.
    if (Lock)
    {
        // Lock specified panel.
        data.LockBits = (1<<PanelNumber);
        data.MaskBits = (1<<PanelNumber);
    }
    else
    {
        // Unlock specified panel.
        data.LockBits = ~(1<<PanelNumber);
        data.MaskBits = (1<<PanelNumber);
    }

    // This overrides the automatic panel lock.
    if ((retvalue = MtcxWriteCommand(MTCX_CMD_KEY_SERVICE, 0, 4, MTCX_DEV_BASEBOARD,
        &data, sizeof(data), &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    return 0;
}
```

Listing 23: SetPanelLock – Panelsperre setzen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.
Die aufgerufene Funktion **MtcxWriteCommand** finden Sie auf Seite 74.

7.4 Versionen lesen

Die folgenden Codebeispiele zeigen, wie Sie die Version des BIOS und der Firmware lesen können, die auf dem PC und auf angeschlossenen Automation Panels installiert ist.

7.4.1 Version des BIOS lesen

Die Version des BIOS kann durch Suche und Auswertung des B&R Vendor Strings im BIOS ROM Speicher gelesen werden.

Das BIOS ROM liegt im Speicher auf Adresse F0000h bis FFFFFh. Der B&R Vendor String ist folgenderweise aufgebaut.

```
„Bernecker + Rainer  Industrie-Elektronik Qv.yy“
„Bernecker + Rainer  Industrie-Elektronik Sv.yy“
```

Wobei Q die Kennung des BIOS für APC910 und PPC900 mit QM77 Chipsatz bzw. S die Kennung des BIOS für einen HM76 Chipsatz ist und v.yy die BIOS Version bezeichnet.

Das folgende C Codebeispiel zeigt, wie die BIOS Kennung und Version gelesen werden können.

```
// Read BIOS ID and version.
//
// Parameters
//   Id
//     [out] Points to a variable that receives the BIOS ID.
//   Version
//     [out] Points to a buffer that receives the
//     BIOS version string: for example "1.14".
//   Size
//     [in] Specifies the size of the version buffer in bytes.
//
// Return 0 at success and -1 at failure.
int GetBiosVersion(char *Id, char *Version, int Size)
{
    const char *BrVendorString = "Bernecker + Rainer  Industrie-Elektronik ";
    unsigned long BiosMem;

    // Search the BIOS memory block for the B&R vendor string.
    for (BiosMem = 0x000F0000; BiosMem < 0x000FFFF0; BiosMem++)
    {
        int i;

        for (i = 0; BrVendorString[i] != '\0'; i++)
        {
            if (READ_REGISTER_UCHAR(BiosMem + i) != BrVendorString[i])
                break;
        }
        if (BrVendorString[i] == '\0')
        {
            BiosMem += i;
            *Id = READ_REGISTER_UCHAR(BiosMem++);
            for (i = 0; i < Size; i++)
                *Version++ = READ_REGISTER_UCHAR(BiosMem++);
            return 0;
        }
    }

    return -1;
}
```

Listing 24: GetBiosVersion – Version des BIOS lesen

7.4.2 Version des MTCX lesen

Die Version der MTCX Firmware kann vom **Version** Register des MTCX (siehe Seite 12) gelesen werden.

Das folgende C Codebeispiel zeigt, wie die MTCX Version gelesen werden kann.

```
// Read version of MTCX.
//
// Parameters
//   Major
//     [out] Points to a variable that receives the
//     major version number: 0 to 255.
//   Minor
//     [out] Points to a variable that receives the
//     minor version number: 0 to 99.
void GetMtcxVersion(unsigned char *Major, unsigned char *Minor)
{
    MTCX_VERSION_REG reg;

    *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_VERSION_ADDR);
    *Major = reg.Major;
    *Minor = reg.Minor;
}
```

Listing 25: GetMtcxVersion – Version des MTCX lesen

Die verwendeten Definitionen finden Sie auf Seite 59.

7.4.3 Version des Display Link FPGA lesen

Die Version der Display Link FPGA Firmware kann aus dem Header der Display Link FPGA Firmware ermittelt werden. Dieser Header kann mit dem **Module Header** Kommando des MTCX (siehe Seite 18) gelesen werden und hat folgenden Aufbau:

```
//
// Strings used in FPGA firmware header.
//
#define APC910_FPGA_ID_LINK "LSDL"

#pragma pack(1)
//
// FPGA firmware header.
//
// Each FPGA firmware has a 16 byte header at address 0x10.
// This header contains the firmware version and update counter.
//
typedef struct
{
    char Id[4];                 // 0x10: ID "LSDL"
    char Version[4];            // 0x14: version: "vvyy", where vv is major and yy is minor
    unsigned long Counter;      // 0x18: update counter; incremented at each update
    unsigned short Reserve2;    // 0x1C: unused; 0
    unsigned short CRC;         // 0x1E: 16-bit CRC of header
} APC910_FPGA_HEADER;
#pragma pack()
```

Listing 26: Datenstruktur APC910_FPGA_HEADER – APC910 FPGA Header

Das folgende C Codebeispiel zeigt, wie die Display Link FPGA Version gelesen werden kann.

```
// Read version of Display Link FPGA.
//
// Parameters
//   Major
//     [out] Points to a variable that receives the
//     major version number: 0 to 255.
//   Minor
//     [out] Points to a variable that receives the
//     minor version number: 0 to 99.
//
// Return 0 at success and -1 at failure.
int GetLinkFpgaVersion(unsigned char *Major, unsigned char *Minor)
{
    APC910_FPGA_HEADER header;
    int retvalue;
    unsigned char ErrorCode;
    unsigned int version;

    // Read active Display Link FPGA firmware header.
    retvalue = MtcxReadCommand(MTCX_CMD_MODULE_HEADER, 0, MTCX_DEVNUM_LINK, 0,
        &header, sizeof(header), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
        // TODO: add your error handling here.
        return -1;
    }

    // Check Display Link FPGA firmware header.
    if (strncmp(header.Id, APC910_FPGA_ID_LINK, strlen(APC910_FPGA_ID_LINK)) != 0)
    {
        // TODO: add your error handling here.
        return -1;
    }

    // Get version from Display Link FPGA firmware header.
    if (sscanf(header.Version, "%4d", &version) != 1)
    {
        // TODO: add your error handling here.
        return -1;
    }

    *Major = version >> 8;
    *Minor = version & 0xff;
    return 0;
}
```

Listing 27: GetLinkFpgaVersion – Version des Display Link FPGA lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.4.4 Version der Scancode Daten lesen

Die Version der Scancode Daten kann mit dem **Version Info** Kommando des MTCX (siehe Seite 18) gelesen werden.

Information:

Die Version der Scancode Daten gibt nur über das Format der Datenstruktur Auskunft und ist unabhängig von der Version, die im B&R Key Editor beim Erstellen der Tastenkonfiguration eingestellt werden kann. Diese Version muss direkt aus den im Flashspeicher abgelegten Daten der Tastenkonfiguration ausgelesen werden (nicht Bestandteil dieser Implementierungsanleitung).

Das folgende C Codebeispiel zeigt, wie die Version der Scancode Daten gelesen werden kann.

```
// Read version of scan code data structure.
//
// Parameters
//   Major
//     [out] Points to a variable that receives the
//     major version number: 0 to 255.
//   Minor
//     [out] Points to a variable that receives the
//     minor version number: 0 to 99.
//
// Return 0 at success and -1 at failure.
//
// Remarks
//   This function returns the supported version of the
//   scan code data structure and NOT the file version!
//   The file version must be read from the Flash memory.
int GetScanCodeVersion(unsigned char *Major, unsigned char *Minor)
{
    MTCX_VERSION_INFO_DATA data;
    int retvalue;
    unsigned char ErrorCode;

    retvalue = MtcxReadCommand(MTCX_CMD_VERSION_INFO, 0, MTCX_DEVNUM_SCAN, MTCX_DEV_BASEBOARD,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
        // TODO: add your error handling here.
        return -1;
    }

    *Major = data.Major;
    *Minor = data.Minor;
    return 0;
}
```

Listing 28: GetScanCodeVersion – Version der Scancode Daten lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.4.5 Version des AP Link FPGA lesen

Die Version des AP Link FPGA eines angeschlossenen Automation Panels kann vom **Panel Version** Register des MTCX (siehe Seite 51) gelesen werden.

Information:

- **Diese Funktion wird nur für Panel 0 bis 14 unterstützt. Die Displayeinheit eines PPC900 (Panel 15) besitzt kein AP Link FPGA.**
- **Für AP800 Extension Units wird die Version der Base Unit geliefert. Ob es sich um eine AP800 Extension Unit handelt, kann über das Panel Version Register (siehe Seite 51 und Codebeispiel auf Seite 92) ermittelt werden.**

Das folgende C Codebeispiel zeigt, wie die Version des AP Link FPGA gelesen werden kann.

```
// Read version of AP link FPGA.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   Major
//     [out] Points to a variable that receives the
//     major version number: 0 to 255.
//   Minor
//     [out] Points to a variable that receives the
//     minor version number: 0 to 99.
//
// Return 0 at success and -1 at failure.
int GetPanelLinkFpgaVersion(unsigned int PanelNumber, unsigned char *Major, unsigned char *Minor)
{
    int retvalue;

    // TODO: acquire synchronization object for panel data here

    retvalue = MtcxSwitchToLinkedPanel(PanelNumber);
    if (retvalue == MTCX_OKAY)
    {
        MTCX_PANEL_VERSION_REG reg;

        *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_PANEL_VERSION_ADDR);
        *Major = reg.Major;
        *Minor = reg.Minor;
    }

    // TODO: release synchronization object for panel data here

    return retvalue;
}
```

Listing 29: GetPanelLinkFpgaVersion – Version des AP Link FPGA lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxSwitchToLinkedPanel** finden Sie auf Seite 68.

7.5 Hardwareeigenschaften lesen

Die folgenden Codebeispiele zeigen, wie Sie Eigenschaften der verwendeten Hardware lesen können.

7.5.1 Gerätetyp (APC910, PPC900) lesen

Der Typ (APC910 oder PPC900) des verwendeten Geräts kann über das **Hardware Info** Register des MTCX (siehe Seite 29) gelesen werden.

Das folgende C Codebeispiel zeigt, wie der Typ des Geräts gelesen werden kann.

```
// Get the type of the device.
//
// Return MTCX device type etc. (see MTCX_DEVICE_* defines).
int GetDeviceType()
{
    MTCX_HW_INFO_REG reg;

    *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_HW_INFO_ADDR);
    return reg.DeviceType;
}
```

Listing 30: GetDeviceType – Gerätetyp (APC910, PPC900) lesen

Die verwendeten Definitionen finden Sie auf Seite 59.

7.5.2 Paneltyp (AP900, AP800, …) lesen

Ob es sich bei einem Panel um ein AP900 oder ein AP800 usw. handelt, kann über das **Panel Version** Register des MTCX (siehe Seite 51) gelesen werden.

Das folgende C Codebeispiel zeigt, wie der Paneltyp gelesen werden kann.

```
// Get the type of a panel.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//
// Return panel type and -1 at failure (e.g. if panel is not linked).
int GetPanelType(unsigned int PanelNumber)
{
    int retvalue;
    int PanelType = -1;

    // TODO: acquire synchronization object for panel data here

    retvalue = MtcxSwitchToLinkedPanel(PanelNumber);
    if (retvalue == MTCX_OKAY)
    {
        MTCX_PANEL_VERSION_REG reg;

        *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_PANEL_VERSION_ADDR);
        PanelType = reg.Id;
    }

    // TODO: release synchronization object for panel data here

    return PanelType;
}
```

Listing 31: GetPanelType – Paneltyp (AP900, AP800, …) lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxSwitchToLinkedPanel** finden Sie auf Seite 68.

7.6 Geräteinformationen des PC lesen

Die folgenden Codebeispiele zeigen, wie Sie die B&R spezifischen Geräteinformationen (z.B. Seriennummer oder Hardwarerevision) lesen können. Damit können Sie auch die Informationen von optionalen Komponenten wie z.B. der USV IF Option lesen.

7.6.1 Gerätekennung eines PC Moduls lesen

Die Gerätekennung eines PC Moduls entspricht der B&R internen Produktnummer und kann mit dem **Module Info** Kommando des MTCX (siehe Seite 29) gelesen werden.

Das folgende C Codebeispiel zeigt, wie die Gerätekennung eines PC Moduls gelesen werden kann.

```
// Get device ID of a module.
//
// Parameters
//   ModuleNumber
//     [in] Specifies the module number: 0 to 15.
//   DeviceId
//     [out] Points to a variable that receives
//     the device ID: 00000000h to FFFFFFFFh.
//
// Return 0 at success and -1 at failure.
int GetModuleDeviceId(int ModuleNumber, unsigned long *DeviceId)
{
    int retvalue;
    unsigned char ErrorCode;

    if ((retvalue = MtcxReadCommand(MTCX_CMD_MODULE_INFO, 0x100, ModuleNumber, 0,
        DeviceId, sizeof(*DeviceId), &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }
    return 0;
}
```

Listing 32: GetModuleDeviceId – Gerätekennung eines PC Moduls lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.6.2 Kompatibilitätskennung eines PC Moduls lesen

Die Kompatibilitätskennung eines PC Moduls dient zur Erkennung von Hardwareänderungen, die für gerätespezifische Software von Bedeutung sind und kann mit dem **Module Info** Kommando des MTCX (siehe Seite 29) gelesen werden.

Das folgende C Codebeispiel zeigt, wie die Kompatibilitätskennung eines PC Moduls gelesen werden kann.

```
// Get compatibility ID of a module.
//
// Parameters
//   ModuleNumber
//     [in] Specifies the module number: 0 to 15.
//   CompatId
//     [out] Points to a variable that receives
//     the compatibility ID: 0000h to FFFFh.
//
// Return 0 at success and -1 at failure.
int GetModuleCompatId(int ModuleNumber, unsigned short *CompatId)
{
#pragma pack(1)
    struct
    {
        unsigned long VendorId;
        unsigned short CompatId;
    } data;
#pragma pack()
    int retvalue;
    unsigned char ErrorCode;

    if ((retvalue = MtcxReadCommand(MTCX_CMD_MODULE_INFO, 0x200, ModuleNumber, 0,
        &data, sizeof(data), &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }
    *CompatId = data.CompatId;
    return 0;
}
```

Listing 33: GetModuleCompatId – Kompatibilitätskennung eines PC Moduls lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.6.3 Herstellerkennung eines PC Moduls lesen

Die Herstellerkennung eines PC Moduls ist bei B&R immer 0 und kann mit dem **Module Info** Kommando des MTCX (siehe Seite 29) gelesen werden.

Das folgende C Codebeispiel zeigt, wie die Herstellerkennung eines PC Moduls gelesen werden kann.

```
// Get vendor ID of a module.
//
// Parameters
//   ModuleNumber
//     [in] Specifies the module number: 0 to 15.
//   VendorId
//     [out] Points to a variable that receives
//     the vendor ID: 00000000h to FFFFFFFFh.
//
// Return 0 at success and -1 at failure.
int GetModuleVendorId(int ModuleNumber, unsigned long *VendorId)
{
#pragma pack(1)
    struct
    {
        unsigned long VendorId;
        unsigned short CompatId;
    } data;
#pragma pack()
    int retvalue;
    unsigned char ErrorCode;

    if ((retvalue = MtcxReadCommand(MTCX_CMD_MODULE_INFO, 0x200, ModuleNumber, 0,
        &data, sizeof(data), &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }
    *VendorId = data.VendorId;
    return 0;
}
```

Listing 34: GetModuleVendorId – Herstellerkennung eines PC Moduls lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.6.4 Hardwarerevision eines PC Moduls lesen

Die Hardwarerevision eines PC Moduls kann mit dem **Module Info** Kommando des MTCX (siehe Seite 29) gelesen werden.

Das folgende C Codebeispiel zeigt, wie die Hardwarerevision eines PC Moduls gelesen werden kann. Geben Sie für HardwareRev eine mindestens 5 char große Variable an.

```
// Get hardware revision of a module.
//
// Parameters
//   ModuleNumber
//     [in] Specifies the module number: 0 to 15.
//   HwRevision
//     [out] Points to a buffer that receives the hardware revision.
//     The buffer must be large enough to hold 4 characters
//     and a 0-byte.
//   Size
//     [in] Specifies the size of the buffer in bytes: minimum is 5.
//
// Return 0 at success and -1 at failure.
int GetModuleHwRevision(int ModuleNumber, char *HwRevision, int Size)
{
    int retvalue;
    unsigned char ErrorCode;

    if (Size < 5)
        return -1;

    if ((retvalue = MtcxReadCommand(MTCX_CMD_MODULE_INFO, 0x300, ModuleNumber, 0,
        HwRevision, 5, &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }
    HwRevision[4] = '\0';
    return 0;
}
```

Listing 35: GetModuleHwRevision – Hardwarerevision eines PC Moduls lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.6.5 Seriennummer eines PC Moduls lesen

Die Seriennummer eines PC Moduls kann mit dem **Module Info** Kommando des MTCX (siehe Seite 29) gelesen werden.

Das folgende C Codebeispiel zeigt, wie die Seriennummer eines PC Moduls gelesen werden kann. Geben Sie für SerialNumber eine mindestens 12 char große Variable an.

```
// Get serial number of a module.
//
// Parameters
//   ModuleNumber
//     [in] Specifies the module number: 0 to 15.
//   SerialNumber
//     [out] Points to a buffer that receives the serial number.
//     The buffer must be large enough to hold 11 characters
//     and a 0-byte.
//   Size
//     [in] Specifies the size of the buffer in bytes: minimum is 12.
//
// Return 0 at success and -1 at failure.
int GetModuleSerialNumber(int ModuleNumber, char *SerialNumber, int Size)
{
    int retvalue;
    unsigned char ErrorCode;

    if (Size < 12)
        return -1;

    if ((retvalue = MtcxReadCommand(MTCX_CMD_MODULE_INFO, 0x400, ModuleNumber, 0,
        SerialNumber, 12, &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }
    SerialNumber[11] = '\0';
    return 0;
}
```

Listing 36: GetModuleSerialNumber – Seriennummer eines PC Moduls lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.6.6 Bestellnummer eines PC Moduls lesen

Die Bestellnummer eines PC Moduls kann mit dem **Module Info** Kommando des MTCX (siehe Seite 29) gelesen werden.

Das folgende C Codebeispiel zeigt, wie die Bestellnummer eines PC Moduls gelesen werden kann. Geben Sie für ModelNumber eine mindestens 41 char große Variable an.

```
// Get model number of a module.
//
// Parameters
//   ModuleNumber
//     [in] Specifies the module number: 0 to 15.
//   ModelNumber
//     [out] Points to a buffer that receives the model number.
//     The buffer must be large enough to hold 40 characters
//     and a 0-byte.
//   Size
//     [in] Specifies the size of the buffer in bytes: minimum is 41.
//
// Return 0 at success and -1 at failure.
int GetModuleModelNumber(int ModuleNumber, char *ModelNumber, int Size)
{
    int retvalue;
    unsigned char ErrorCode;

    if (Size < 41)
        return -1;

    if ((retvalue = MtcxReadCommand(MTCX_CMD_MODULE_INFO, 0x500, ModuleNumber, 0,
        ModelNumber, 16, &ErrorCode)) != MTCX_OKAY ||
        (retvalue = MtcxReadCommand(MTCX_CMD_MODULE_INFO, 0x501, ModuleNumber, 0,
        ModelNumber+16, 16, &ErrorCode)) != MTCX_OKAY ||
        (retvalue = MtcxReadCommand(MTCX_CMD_MODULE_INFO, 0x502, ModuleNumber, 0,
        ModelNumber+32, 9, &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }
    ModelNumber[40] = '\0';
    return 0;
}
```

Listing 37: GetModuleModelNumber – Bestellnummer eines PC Moduls lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.6.7 Parent Gerätekennung eines PC Moduls lesen

Die Parent Gerätekennung eines PC Moduls gibt die Gerätekennung des PC Moduls an, von dem das aktuelle Gerät „abgeleitet“ wurde (z.B. bei einer kundenspezifischen Variante eines Standardgeräts). Die Parent Gerätekennung kann mit dem **Module Info** Kommando des MTCX (siehe Seite 29) gelesen werden.

Das folgende C Codebeispiel zeigt, wie die Parent Gerätekennung eines PC Moduls gelesen werden kann. ID = FFFFh zeigt an, dass kein Parent Gerät existiert.

```
// Get parent device ID of a module.
//
// Parameters
//   ModuleNumber
//     [in] Specifies the module number: 0 to 15.
//   DeviceId
//     [out] Points to a variable that receives
//     the device ID: 00000000h to FFFFFFFFh (= no parent).
//
// Return 0 at success and -1 at failure.
int GetModuleParentDeviceId(int ModuleNumber, unsigned long *DeviceId)
{
#pragma pack(1)
     struct
     {
            unsigned long DeviceId;
            unsigned short CompatId;
     } data;
#pragma pack()
    int retvalue;
    unsigned char ErrorCode;

    if ((retvalue = MtcxReadCommand(MTCX_CMD_MODULE_INFO, 0x700, ModuleNumber, 0,
        &data, sizeof(data), &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }
    *DeviceId = data.DeviceId;
    return 0;
}
```

Listing 38: GetModuleParentDeviceId – Parent Gerätekennung eines PC Moduls lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.6.8 Parent Kompatibilitätskennung eines PC Moduls lesen

Die Parent Kompatibilitätskennung eines PC Moduls gibt die Kompatibilitätskennung des PC Moduls an, von dem das aktuelle Gerät „abgeleitet“ wurde (z.B. bei einer kundenspezifischen Variante eines „Standard“ Geräts). Die Parent Kompatibilitätskennung kann mit dem **Module Info** Kommando des MTCX (siehe Seite 29) gelesen werden.

Das folgende C Codebeispiel zeigt, wie die Parent Kompatibilitätskennung eines PC Moduls gelesen werden kann. ID = FFFFh zeigt an, dass kein Parent Gerät existiert.

```
// Get parent compatibility ID of a module.
//
// Parameters
//   ModuleNumber
//     [in] Specifies the module number: 0 to 15.
//   CompatId
//     [out] Points to a variable that receives
//     the compatibility ID: 0000h to FFFFh (= no parent).
//
// Return 0 at success and -1 at failure.
int GetModuleParentCompatId(int ModuleNumber, unsigned short *CompatId)
{
#pragma pack(1)
    struct
    {
        unsigned long DeviceId;
        unsigned short CompatId;
    } data;
#pragma pack()
    int retvalue;
    unsigned char ErrorCode;

    if ((retvalue = MtcxReadCommand(MTCX_CMD_MODULE_INFO, 0x700, ModuleNumber, 0,
        &data, sizeof(data), &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }
    *CompatId = data.CompatId;
    return 0;
}
```

Listing 39: GetModuleParentCompatId – Parent Kompatibilitätskennung eines PC Moduls lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.7 Geräteinformationen eines Panels lesen

Die folgenden Codebeispiele zeigen, wie Sie die B&R spezifischen Geräteinformationen (z.B. Seriennummer oder Hardwarerevision) eines angeschlossenen Automation Panels lesen können.

Die in den folgenden Codebeispielen gelesenen Informationen sind Bestandteil der Factory Settings des Automation Panels und liegen in folgender Datenstruktur vor (Startoffset in den Factory Settings ist 80h):

```
#define BR_INT_DATA_PANEL_HEADER_ID "\xff\x55\xaa\xff\x00\x00\xff\xff"
#define BR_INT_DATA_PANEL_HEADER_SIZE 8

#pragma pack(1)

typedef struct
{
    unsigned char   Header[8];              // hex: FF 55 AA FF 00 00 FF
    unsigned char   Reserve1[2];
    unsigned long   DeviceId;               // old material number (big-endian format!)
    unsigned short  CompatId;               // compatibility ID (big-endian format!)
    unsigned long   VendorId;               // B&R = 0 (big-endian format!)
    char            HwRevision[5];          // string with 4 char's
    char            SerialNumber[12];       // string with 11 char's
    char            ModelNumber[41];        // string with 40 char's
    unsigned char   Reserve2[2];
} BR_INT_DATA_PANEL; // start offset 80h
```

Listing 40: Datenstruktur BR_INT_DATA_PANEL - Geräteinformationen eines Panels

Information:

Die Factory Settings der Displayeinheit eines PPC900 (Panel 15) können über das Module Info Kommando (siehe Seite 29 oder Codebeispiele ab Seite 93) gelesen werden. AP800 Extension Units besitzen keine eigenen Factory Settings.

Diese Daten können mit der nächsten Funktion gelesen werden:

```
// Read internal data of panel.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   Data
//     [out] Points to a buffer that receives the B&R internal data.
//
// Return 0 at success and -1 at failure.
int ReadPanelIntData(unsigned int PanelNumber, BR_INT_DATA_PANEL *Data)
{
    int retvalue;
    MTCX_PANEL_VERSION_REG reg;
    int i;

    if (PanelNumber > 15)
        return -1;
    if (sizeof(BR_INT_DATA_PANEL) % MTCX_CMD_DATA_SIZE) // safety check
        return -1;

    // TODO: acquire synchronization object for panel data here

    retvalue = MtcxSwitchToLinkedPanel(PanelNumber);
    if (retvalue == MTCX_OKAY)
        *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_PANEL_VERSION_ADDR);

    // TODO: release synchronization object for panel data here

    if (retvalue != MTCX_OKAY)
        return -1;
```

```
    // Don't send Flash command to built-in display because MTCX does not
    // respond on some platforms...
    if (reg.Id == MTCX_VER_ID_DISP)
       return -1;

    // Read B&R internal data from flash memory.
    for (i = 0; i < sizeof(BR_INT_DATA_PANEL); i += MTCX_CMD_DATA_SIZE)
    {
        unsigned char ErrorCode;

        retvalue = MtcxReadCommand(MTCX_CMD_FLASH_SERVICE, (0x80 + i) / MTCX_CMD_DATA_SIZE,
            PanelNumber, MTCX_DEV_PANELSET, (unsigned char*)Data + i, MTCX_CMD_DATA_SIZE,
&ErrorCode);
        if (retvalue != MTCX_OKAY)
        {
            // TODO: add your error handling here
            return -1;
        }
    }

    if (memcmp(Data->Header, BR_INT_DATA_PANEL_HEADER_ID, BR_INT_DATA_PANEL_HEADER_SIZE) != 0)
    {
        // the flash memory does not contain valid data
        return -1;
    }

    return 0;
}
```

Listing 41: ReadPanelIntData – B&R interne Daten eines Panels lesen

Information:

Das Lesen der Factory Settings Daten dauert aufgrund der notwendigen Flash / EEPROM Zugriffe länger als andere MTCX Kommandos.

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.7.1 Gerätekennung eines Panels lesen

Die Gerätekennung eines angeschlossenen Automation Panels entspricht der B&R internen Produktnummer und kann aus den Factory Settings des Panels mit dem **Flash / EEPROM Service** Kommando (siehe Seite 24) des MTCX gelesen werden.

Das folgende C Codebeispiel zeigt, wie die Gerätekennung eines angeschlossenen Automation Panels gelesen werden kann.

```
// Get device ID of panel.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   DeviceId
//     [out] Points to a variable that receives
//     the device ID: 00000000h to FFFFFFFFh.
//
// Return 0 at success and -1 at failure.
int GetPanelDeviceId(unsigned int PanelNumber, unsigned long *DeviceId)
{
    BR_INT_DATA_PANEL data;

    if (ReadPanelIntData(PanelNumber, &data) != 0)
        return -1;

    *DeviceId = swapl(data.DeviceId);
    return 0;
}
```

Listing 42: GetPanelDeviceId – Gerätekennung eines Panels lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **ReadPanelIntData** finden Sie auf Seite 101.
Die aufgerufene Funktion **swapl** finden Sie auf Seite 58.

7.7.2 Kompatibilitätskennung eines Panels lesen

Die Kompatibilitätskennung eines angeschlossenen Automation Panels kann aus den Factory Settings des Panels mit dem **Flash / EEPROM Service** Kommando des MTCX (siehe Seite 24) gelesen werden.

Das folgende C Codebeispiel zeigt, wie die Kompatibilitätskennung eines angeschlossenen Automation Panels gelesen werden kann.

```
// Get compatibility ID of panel.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   CompatId
//     [out] Points to a variable that receives
//     the compatibility ID: 0000h to FFFFh.
//
// Return 0 at success and -1 at failure.
int GetPanelCompatId(unsigned int PanelNumber, unsigned short *CompatId)
{
    BR_INT_DATA_PANEL data;

    if (ReadPanelIntData(PanelNumber, &data) != 0)
        return -1;

    *CompatId = swapw(data.CompatId);
    return 0;
}
```

Listing 43: GetPanelCompatId – Kompatibilitätskennung eines Panels lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **ReadPanelIntData** finden Sie auf Seite 101.
Die aufgerufene Funktion **swapw** finden Sie auf Seite 58.

7.7.3 Herstellerkennung eines Panels lesen

Die Herstellerkennung eines angeschlossenen Automation Panels kann aus den Factory Settings des Panels mit dem **Flash / EEPROM Service** Kommando des MTCX (siehe Seite 24) gelesen werden.

Das folgende C Codebeispiel zeigt, wie die Herstellerkennung eines angeschlossenen Automation Panels gelesen werden kann.

```
// Get vendor ID of panel.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   VendorId
//     [out] Points to a variable that receives
//     the vendor ID: 00000000h to FFFFFFFFh.
//
// Return 0 at success and -1 at failure.
int GetPanelVendorId(unsigned int PanelNumber, unsigned long *VendorId)
{
    BR_INT_DATA_PANEL data;

    if (ReadPanelIntData(PanelNumber, &data) != 0)
        return -1;

    *VendorId = swapl(data.VendorId);
    return 0;
}
```

Listing 44: GetPanelVendorId – Herstellerkennung eines Panels lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **ReadPanelIntData** finden Sie auf Seite 101.
Die aufgerufene Funktion **swapl** finden Sie auf Seite 58.

7.7.4 Hardwarerevision eines Panels lesen

Die Hardwarerevision eines angeschlossenen Automation Panels kann aus den Factory Settings des Panels mit dem **Flash / EEPROM Service** Kommando des MTCX (siehe Seite 24) gelesen werden.

Das folgende C Codebeispiel zeigt, wie die Hardwarerevision eines angeschlossenen Automation Panels gelesen werden kann. Geben Sie für HardwareRev eine mindestens 5 char große Variable an.

```
// Get hardware revision of panel.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   HwRevision
//     [out] Points to a buffer that receives the hardware revision.
//     The buffer must be large enough to hold 4 characters
//     and a 0-byte.
//   Size
//     [in] Specifies the size of the buffer in bytes: minimum is 5.
//
// Return 0 at success and -1 at failure.
int GetPanelHwRevision(unsigned int PanelNumber, char *HwRevision, int Size)
{
    BR_INT_DATA_PANEL data;

    if (Size < 5)
        return -1;

    if (ReadPanelIntData(PanelNumber, &data) != 0)
        return -1;

    strncpy(HwRevision, data.HwRevision, Size);
    HwRevision[4] = '\0';
    return 0;
}
```

Listing 45: GetPanelHwRevision – Hardwarerevision eines Panels lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **ReadPanelIntData** finden Sie auf Seite 101.

7.7.5 Seriennummer eines Panels lesen

Die Seriennummer eines angeschlossenen Automation Panels kann aus den Factory Settings des Panels mit dem **Flash / EEPROM Service** Kommando des MTCX (siehe Seite 24) gelesen werden.

Das folgende C Codebeispiel zeigt, wie die Seriennummer eines angeschlossenen Automation Panels gelesen werden kann. Geben Sie für SerialNumber eine mindestens 12 char große Variable an.

```
// Get serial number of panel.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   HwRevision
//     [out] Points to a buffer that receives the hardware revision.
//     The buffer must be large enough to hold 11 characters
//     and a 0-byte.
//   Size
//     [in] Specifies the size of the buffer in bytes: minimum is 12.
//
// Return 0 at success and -1 at failure.
int GetPanelSerialNumber(unsigned int PanelNumber, char *SerialNumber, int Size)
{
    BR_INT_DATA_PANEL data;

    if (Size < 12)
        return -1;

    if (ReadPanelIntData(PanelNumber, &data) != 0)
        return -1;

    strncpy(SerialNumber, data.SerialNumber, 12);
    SerialNumber[11] = '\0';
    return 0;
}
```

Listing 46: GetPanelSerialNumber – Seriennummer eines Panels lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **ReadPanelIntData** finden Sie auf Seite 101.

7.7.6 Bestellnummer eines Panels lesen

Die Bestellnummer eines angeschlossenen Automation Panels kann aus den Factory Settings des Panels mit dem **Flash / EEPROM Service** Kommando des MTCX (siehe Seite 24) gelesen werden.

Das folgende C Codebeispiel zeigt, wie die Bestellnummer eines angeschlossenen Automation Panels gelesen werden kann. Geben Sie für ModelNumber eine mindestens 41 char große Variable an.

```
// Get model number of panel.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   ModelNumber
//     [out] Points to a buffer that receives the model number.
//     The buffer must be large enough to hold 40 characters
//     and a 0-byte.
//   Size
//     [in] Specifies the size of the buffer in bytes: minimum is 41.
//
// Return 0 at success and -1 at failure.
int GetPanelModelNumber(unsigned int PanelNumber, char *ModelNumber, int Size)
{
    BR_INT_DATA_PANEL data;

   if (Size < 41)
        return -1;

    if (ReadPanelIntData(PanelNumber, &data) != 0)
        return -1;

    strncpy(ModelNumber, data.ModelNumber, 41);
    ModelNumber[40] = '\0';
    return 0;
}
```

Listing 47: GetPanelModelNumber – Bestellnummer eines Panels lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **ReadPanelIntData** finden Sie auf Seite 101.

7.8 Statistikwerte lesen

Sie können folgende Statistikwerte lesen:

- Betriebsstunden und Einschaltzyklen eines PC Moduls
- Betriebsstunden und Einschaltzyklen (der Hintergrundbeleuchtung) der Displayeinheit eines PPC900 bzw. eines Automation Panels
- Betriebsstunden und Einschaltzyklen der PC Lüfter
- CMOS Batteriezustand

Die Statistikwerte auf dem PC werden vom MTCX verwaltet. Auf einem Automation Panel werden die Statistikwerte vom Panel selbst aktualisiert.

Die folgenden Codebeispiele zeigen, wie Sie die Statistikwerte (z.B. Betriebsstundenzähler) der PC Module und von angeschlossenen Automation Panels lesen können.

7.8.1 Betriebsstunden und Einschaltzyklen eines PC Moduls lesen

Die Betriebsstunden und Einschaltzyklen eines PC Moduls können mit dem **Module Statistics** Kommando des MTCX (siehe Seite 32) gelesen werden.

Das folgende C Codebeispiel zeigt, wie die Betriebsstunden und Einschaltzyklen eines PC Moduls gelesen werden können.

```
// Read statistics values of a module.
//
// Parameters
//   ModuleNumber
//     [in] Specifies the module number: 0 to 15.
//   PowerOnHours
//     [out] Points to a variable that receives the
//     power-on hours: 0 to 4194303.
//   PowerOnCycles
//     [out] Points to a variable that receives the
//     power-on cycles: 0 to 16777212.
//
// Return 0 at success and -1 at failure.
int GetModuleStatistics(int ModuleNumber, unsigned long *PowerOnHours, unsigned long
*PowerOnCycles)
{
    MTCX_MODULE_STATISTICS_DATA data;
    int retvalue;
    unsigned char ErrorCode;

    retvalue = MtcxReadCommand(MTCX_CMD_MODULE_STATISTICS, 0x100, ModuleNumber, 0,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    *PowerOnHours = data.PowerOnHours / 4;
    *PowerOnCycles = data.PowerOnCycles;
}
```

Listing 48: GetModuleStatistics – Betriebsstunden und Einschaltzyklen eines PC Moduls lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.8.2 Betriebsstunden und Einschaltzyklen der Displayeinheit lesen

Die Betriebsstunden und Einschaltzyklen (der Hintergrundbeleuchtung) der Displayeinheit eines PPC900 können mit dem **Module Statistics** Kommando des MTCX (siehe Seite 32) gelesen werden.

Das folgende C Codebeispiel zeigt, wie die Betriebsstunden und Einschaltzyklen der Displayeinheit gelesen werden können.

```
// Read statistics values of the display unit (PPC900 only).
//
// Parameters
//   ModuleNumber
//     [in] Specifies the module number: 1.
//   BacklightOnHours
//     [out] Points to a variable that receives the
//     backlight-on hours: 0 to 4194303.
//   BacklightOnCycles
//     [out] Points to a variable that receives the
//     backlight-on cycles: 0 to 16777212.
//
// Return 0 at success and -1 at failure.
int GetDisplayStatistics(int ModuleNumber, unsigned long *BacklightOnHours, unsigned long
*BacklightOnCycles)
{
    MTCX_DISPLAY_STATISTICS_DATA data;
    int retvalue;
    unsigned char ErrorCode;

    retvalue = MtcxReadCommand(MTCX_CMD_MODULE_STATISTICS, 0x100, ModuleNumber, 1,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    *BacklightOnHours = data.BacklightOnHours / 4;
    *BacklightOnCycles = data.BacklightOnCycles;
    return 0;
}
```

Listing 49: GetDisplayStatistics – Betriebsstunden und Einschaltzyklen der Displayeinheit lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.8.3 Betriebsstunden und Einschaltzyklen der PC Lüfter lesen

Die Betriebsstunden und Einschaltzyklen der PC Lüfter können mit dem **Module Statistics** Kommando des MTCX (siehe Seite 32) gelesen werden.

Jedes PC Modul kann bis zu vier Lüfter besitzen. Das Kommando liefert für nicht vorhandene Lüfter einen Fehler.

Hinweis: Normalerweise liefern alle Fans die gleichen Statistikwerte – außer im Falle eines Lüfterfehlers (z.B. Stillstand).

Das folgende C Codebeispiel zeigt, wie die Betriebsstunden und Einschaltzyklen eines PC Lüfters eines Moduls gelesen werden können.

```
// Read statistics values of the fans on a module.
//
// Parameters
//   ModuleNumber
//     [in] Specifies the module number: 8 or 12 (PPC900 only).
//   FanNumber
//     [in] Specifies the fan number: 0 to 3.
//   FanOnHours
//     [out] Points to a variable that receives the
//     fan operating hours: 0 to 4194303.
//   FanOnCycles
//     [out] Points to a variable that receives the
//     fan turn-on cycles: 0 to 16777212.
//
// Return 0 at success and -1 at failure.
int GetFanStatistics(int ModuleNumber, int FanNumber, unsigned long *FanOnHours, unsigned long
*FanOnCycles)
{
    MTCX_FAN_STATISTICS_DATA data;
    int retvalue;
    unsigned char ErrorCode;

    retvalue = MtcxReadCommand(MTCX_CMD_MODULE_STATISTICS, 0x200, ModuleNumber, FanNumber,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    *FanOnHours = data.FanOnHours / 4;
    *FanOnCycles = data.FanOnCycles;
    return 0;
}
```

Listing 50: GetFanStatistics – Betriebsstunden und Einschaltzyklen der PC Lüfter lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.8.4 On-Battery Stunden und Zyklen der USV lesen

Die „On-Battery“ Stunden und Zyklen der USV können mit dem **Module Statistics** Kommando des MTCX (siehe Seite 32) gelesen werden.

Das folgende C Codebeispiel zeigt, wie die „On-Battery“ Stunden und Zyklen der USV gelesen werden können.

```
// Read statistics values of the UPS (battery).
//
// Parameters
//   OnBatHours
//     [out] Points to a variable that receives the
//     "on battery" hours: 0 to 4194303.
//   OnBatCycles
//     [out] Points to a variable that receives the
//     "on battery" cycles: 0 to 16777212.
//
// Return 0 at success and -1 at failure.
int GetUpsStatistics(unsigned long *OnBatHours, unsigned long *OnBatCycles)
{
    MTCX_UPS_STATISTICS_DATA data;
    int retvalue;
    unsigned char ErrorCode;

    retvalue = MtcxReadCommand(MTCX_CMD_MODULE_STATISTICS, 0x100, MTCX_DEVNUM_APC910_IF1, 3,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    *OnBatHours = data.OnBatHours / 4;
    *OnBatCycles = data.OnBatCycles;
    return 0;
}
```

Listing 51: GetUpsStatistics – „On Battery“ Stunden und Zyklen der USV lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.8.5 Betriebsstunden und Einschaltzyklen eines Panels lesen

Die Betriebsstunden und Einschaltzyklen (der Hintergrundbeleuchtung) eines angeschlossenen Automation Panels können mit dem **Statistics Info** Kommando des MTCX (siehe Seite 25) gelesen werden.

Information:

- **Diese Funktion wird nur für Panel 0 bis 14 unterstützt. Die Statistikwerte der Displayeinheit eines PPC900 (Panel 15) können über das Module Statistics Kommando (siehe Seite 32 und Codebeispiel auf Seite 109) gelesen werden.**
- **Diese Funktion wird für AP800 Extension Units nicht unterstützt. Ob es sich um eine AP800 Extension Unit handelt, kann über das Panel Version Register (siehe Seite 51 und Codebeispiel auf Seite 92) ermittelt werden.**

Das folgende C Codebeispiel zeigt, wie die Betriebsstunden und Einschaltzyklen eines angeschlossenen Automation Panels gelesen werden können.

```
// Get statistics values of a panel.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   BacklightOnHours
//     [out] Points to a variable that receives the
//     backlight-on hours: 0 to 65535.
//   BacklightOnCycles
//     [out] Points to a variable that receives the
//     backlight-on cycles: 0 to 65535.
//
// Return 0 at success and -1 at failure.
int GetPanelStatistics(unsigned int PanelNumber, unsigned short *BacklightOnHours, unsigned short
*BacklightOnCycles)
{
    MTCX_PANEL_STATISTICS_DATA data;
    unsigned char ErrorCode;
    int retvalue;

    // Check if panel is linked.
    retvalue = MtcxSwitchToLinkedPanel(PanelNumber);
    if (retvalue != MTCX_OKAY)
    {
         // TODO: add your error handling here
         return -1;
    }

    if ((retvalue = MtcxReadCommand(MTCX_CMD_STATISTICS_INFO, 0, PanelNumber, MTCX_DEV_PANEL,
        &data, sizeof(data), &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    *BacklightOnHours = data.BacklightOnHours;
    *BacklightOnCycles = data.BacklightOnCycles;
    return 0;
}
```

Listing 52: GetPanelStatistics – Betriebsstunden und Einschaltzyklen eines Panels lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.8.6 CMOS Batteriezustand lesen

Der CMOS Batteriezustand kann über das **Baseboard Specials** Register des MTCX (siehe Seite 45) gelesen werden. Der Zustand der Batterie wird als Integer Wert geliefert: 0 = gut, 3 = schlecht

Das folgende C Codebeispiel zeigt, wie der CMOS Batteriezustand gelesen werden kann.

```
// Read CMOS battery state.
//
// Parameters
//   State
//     [out] Points to a variable that receives the
//     state of the CMOS battery:
//     0 = good,
//     2 = not available (older platforms only),
//     3 = bad.
void GetBatteryState(int* State)
{
    MTCX_BASEBOARD_SPECIALS_REG reg;

    *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_BASEBOARD_SPECIALS_ADDR);
    *State = reg.BatState;
}
```

Listing 53: GetBatteryState – CMOS Batteriezustand lesen

Die verwendeten Definitionen finden Sie auf Seite 59.

7.9 Temperaturen lesen

Sie können folgende Temperaturen auslesen:
- Temperaturen eines PC Moduls
- Temperatur eines angeschlossenen Automation Panel
- Temperatur der USV Batterie (siehe Seite 154)

Alle Temperaturen werden vom MTCX erfasst und zur Verfügung gestellt.

Die folgenden Codebeispiele zeigen, wie Sie die Temperaturen eines PC Moduls und von angeschlossenen Automation Panels lesen können.

7.9.1 Temperaturen eines PC Moduls lesen

Die Temperaturen eines PC Moduls können über das **Module Temperature** Kommando des MTCX (siehe Seite 31) gelesen werden.

Jedes PC Modul kann bis zu vier Temperatursensoren besitzen. Das Kommando liefert für nicht vorhandene Temperatursensoren einen Fehler.

Das folgende C Codebeispiel zeigt, wie die Temperatur eines PC Moduls gelesen werden kann.

```
// Read temperature from a module.
//
// Parameters
//   ModuleNumber
//     [in] Specifies the module number: 0 to 15.
//   SensorNumber
//     [in] Specifies the sensor number: 0 to 3.
//   Temp
//     [out] Points to a variable that receives
//     the temperature in degrees Celsius.
//
// Return 0 at success and -1 at failure.
//
// Remarks: See to the hardware manual for a
// description of the temperature sensors.
int GetModuleTemp(int ModuleNumber, int SensorNumber, float *Temp)
{
    MTCX_MODULE_TEMPERATURE_DATA data;
    int retvalue;
    unsigned char ErrorCode;

    retvalue = MtcxReadCommand(MTCX_CMD_MODULE_TEMPERATURE, 0, ModuleNumber, SensorNumber,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    *Temp = (float)(data.Temp >> 6) / 4;
    return 0;
}
```

Listing 54: GetModuleTemp – Temperatur eines PC Moduls lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.9.2 Temperatur eines Panels lesen

Die Temperatur eines angeschlossenen Automation Panels (Umgebungstemperatur des Displays) kann über das **Panel TempFan** Register des MTCX (siehe Seite 51) gelesen werden.

Information:

- **Diese Funktion wird nur für Panel 0 bis 14 unterstützt. Die Temperatur der Displayeinheit eines PPC900 (Panel 15) wird mit dieser Funktion immer mit 0 geliefert und kann über das Module Temperature Kommando (siehe Seite 31 und Codebeispiel auf Seite 115) gelesen werden.**
- **Für AP800 Extension Units wird keine gültige Temperatur geliefert. Ob es sich um eine AP800 Extension Unit handelt, kann über das Panel Version Register (siehe Seite 51 und Codebeispiel auf Seite 92) ermittelt werden.**

Das folgende C Codebeispiel zeigt, wie die Temperatur eines angeschlossenen Automation Panels gelesen werden kann.

Aus Performancegründen wird in **MtcxGetPanelTemp** nur das niederwertigste Byte des **Panel TempFan** Registers angesprochen.

```
// Read panel temperature.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   Temp
//     [out] Points to variable that receives the
//     panel temperature in degrees Celsius: 0 to 127.
//
// Return 0 at success and -1 at failure.
int GetPanelTemp(unsigned int PanelNumber, char *Temp)
{
    int retvalue;

    // TODO: acquire synchronization object for panel data here

    retvalue = MtcxSwitchToLinkedPanel(PanelNumber);
    if (retvalue == MTCX_OKAY)
        *Temp = (char)READ_PORT_UCHAR(MTCX_PANEL_TEMPFAN_ADDR); // read byte 0 only

    // TODO: release synchronization object for panel data here

    if (retvalue != MTCX_OKAY)
        return -1;
    return 0;
}
```

Listing 55: GetPanelTemp – Panel Temperatur lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxSwitchToLinkedPanel** finden Sie auf Seite 68.

7.10 Lüfterdrehzahlen lesen

Die folgenden Codebeispiele zeigen, wie Sie die Drehzahlen der Lüfter lesen können.

7.10.1 Drehzahl der PC Lüfter lesen

Die Drehzahl der Lüfter eines PC Moduls kann mit dem **Module Fan** Kommando des MTCX (siehe Seite 38) gelesen werden.

Jedes PC Modul kann bis zu vier Lüfter besitzen. Das Kommando liefert für nicht vorhandene Lüfter einen Fehler.

Das folgende C Codebeispiel zeigt, wie die Lüfterdrehzahl eines PC Moduls gelesen werden kann.

```
// Read fan speed from a module.
//
// Parameters
//   ModuleNumber
//     [in] Specifies the module number: 0 to 15.
//   FanNumber
//     [in] Specifies the fan number: 0 to 3.
//   FanSpeed
//     [out] Points to a variable that receives
//     the fan speed in rpm.
//
// Return 0 at success and -1 at failure.
int GetModuleFanSpeed(int ModuleNumber, int FanNumber, unsigned short *FanSpeed)
{
    MTCX_MODULE_FAN_DATA data;
    int retvalue;
    unsigned char ErrorCode;

    retvalue = MtcxReadCommand(MTCX_CMD_MODULE_FAN, 0, ModuleNumber, FanNumber,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    *FanSpeed = data.FanSpeed;
    return 0;
}
```

Listing 56: GetModuleFanSpeed – Lüfterdrehzahl eines PC Moduls lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.10.2 Lüfterdrehzahl des Panels lesen

Die Lüfterdrehzahl eines angeschlossenen Automation Panels kann über das **Panel TempFan** Register des MTCX (siehe Seite 51) gelesen werden.

Information:

- **Diese Funktion wird nur für Panel 0 bis 14 unterstützt. Die Lüfterdrehzahl der Displayeinheit eines PPC900 (Panel 15) wird mit dieser Funktion immer mit 0 geliefert.**
- **Für AP800 Extension Units wird keine gültige Lüfterdrehzahl geliefert. Ob es sich um eine AP800 Extension Unit handelt, kann über das Panel Version Register (siehe Seite 51 und Codebeispiel auf Seite 92) ermittelt werden.**

Das folgende C Codebeispiel zeigt, wie die Lüfterdrehzahl eines angeschlossenen Automation Panels gelesen werden kann.

```
// Read panel fan speed.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   FanSpeed
//     [out] Points to a variable that receives the
//     fan speed in rpm: 0 to 16380.
//
// Return 0 at success and -1 at failure.
int GetPanelFanSpeed(unsigned int PanelNumber, unsigned short *FanSpeed)
{
    int retvalue;

    // TODO: acquire synchronization object for panel data here

    retvalue = MtcxSwitchToLinkedPanel(PanelNumber);
    if (retvalue == MTCX_OKAY)
    {
        MTCX_PANEL_TEMPFAN_REG reg;

        *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_PANEL_TEMPFAN_ADDR);
        *FanSpeed = (unsigned short)reg.FanSpeed * 4;
    }

    // TODO: release synchronization object for panel data here

    if (retvalue != MTCX_OKAY)
        return -1;
    return 0;
}
```

Listing 57: GetPanelFanSpeed – Lüfterdrehzahl des Panels lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxSwitchToLinkedPanel** finden Sie auf Seite 68.

7.11 Spannungswerte lesen

Sie können folgende Spannungen auslesen:

- Batteriespannung Vbat
- Eingangsspannung Vin
- sowie die aktuelle Systemleistung

Alle Werte werden vom MTCX erfasst und zur Verfügung gestellt.

Die folgenden Codebeispiele zeigen, wie Sie die Spannungen lesen können.

7.11.1 Spannungswerte eines PC Moduls lesen

Die Spannungswerte eines PC Moduls können über das **Module Voltage** Kommando des MTCX (siehe Seite 36) gelesen werden.

Hinweis: Aktuell können Spannungswerte nur für die Modulnummer 0 (Systemeinheit) gelesen werden.

Information:

Für das Lesen von gültiger Eingangsspannung (Vin) und Systemleistung (Wcpu) ist MTCX Version 1.01 oder höher auf APC910 und MTCX Version 1.02 oder höher auf PPC900 notwendig. Ältere MTCX Versionen liefern ungültige Werte. Vin ist immer 0.

Das folgende C Codebeispiel zeigt, wie die Spannungswerte gelesen werden können.

```
// Get voltages.
//
// Parameters
//   Vbat
//     [out] Points to a variable that receives the
//     battery voltage in Volt.
//     Note: Vbat returns the minimum value that
//           can be measured when VbatState = bad.
//   VbatState
//     [out] Points to a variable that receives the
//     battery voltage status: 0 = good, 1 = bad.
//   Vin
//     [out] Points to a variable that receives the
//     input voltage in Volt.
//   Wcpu
//     [out] Points to a variable that receives the
//     system power consumption in Watt.
//
// Return 0 at success and -1 at failure.
int GetVoltages(float *Vbat, int *VbatState, float *Vin, float *Wcpu)
{
    MTCX_MODULE_VOLTAGE_DATA data;
    unsigned char ErrorCode;
    int retvalue;

    if ((retvalue = MtcxReadCommand(MTCX_CMD_MODULE_VOLTAGE, 0, 0, 0,
        &data, sizeof(data), &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    *Vbat = (float)data.Vbat.Value / 100;
    *VbatState = data.Vbat.State;
    *Vin = (float)data.Vin.Value / 10;
    *Wcpu = (float)data.Wcpu / 100;
    return 0;
}
```

Listing 58: GetVoltages – Spannungswerte lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.

7.12 Displayfunktionen

An einem APC910 und PPC900 können ein oder mehrere Automation Panels angeschlossen werden.
Für das Display eines Automation Panels können Sie folgende Funktionen ausführen:

- Displayhelligkeit lesen und einstellen
- Display Equalizerwert lesen und einstellen

Für die Displayeinheit eines PPC900 können Sie folgende Funktionen ausführen:

- Displayhelligkeit lesen und einstellen

Die Displayfunktionen werden vom MTCX ausgeführt.

Die folgenden Codebeispiele zeigen, wie Sie die Displayfunktionen ausführen können.

7.12.1 Displayhelligkeit lesen/einstellen

Die Displayhelligkeit (der Hintergrundbeleuchtung) der Displayeinheit eines PPC900 und eines angeschlossenen Automation Panels kann über das **Display Service** Kommando des MTCX (siehe Seite 27) gelesen und eingestellt werden.

Die eingestellte Helligkeit gilt nur bis zum nächsten Start des Systems. Wenn die Helligkeit auch nach dem nächsten Systemstart verwendet werden soll, müssen Sie diese in Ihrer Anwendung speichern und beim nächsten Systemstart nochmals einstellen.

Das folgende C Codebeispiel zeigt, wie die Helligkeit eines Displays gelesen werden kann.

```
// Get display brightness (backlight).
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   Percent
//     [out] Points to a variable that receives the
//     brightness in percent: 0 to 100.
//
// Return 0 at success and -1 at failure.
int GetDisplayBrightness(unsigned int PanelNumber, unsigned char *Percent)
{
    MTCX_DISPLAY_SERVICE_DATA data;
    int retvalue;
    unsigned char ErrorCode;

    // Check if panel is linked.
    retvalue = MtcxSwitchToLinkedPanel(PanelNumber);
    if (retvalue != MTCX_OKAY)
    {
         // TODO: add your error handling here
         return -1;
    }

    // Read display settings.
    retvalue = MtcxReadCommand(MTCX_CMD_DISPLAY_SERVICE, 0x00, PanelNumber, MTCX_DEV_PANEL,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
         // TODO: add your error handling here
         return -1;
    }

    // Brightness not supported?
    if (data.Brightness == 255)
        return -1;

    *Percent = data.Brightness;
    return 0;
}
```

Listing 59: GetDisplayBrightness – Displayhelligkeit lesen

Das folgende C Codebeispiel zeigt, wie die Helligkeit eines Displays eingestellt werden kann (auch wenn das Panel nicht angeschlossen ist).

```
// Set display brightness (backlight).
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   Percent
//     [in] Specifies the brightness in percent: 0 to 100.
//
// Return 0 at success and -1 at failure.
int SetDisplayBrightness(unsigned int PanelNumber, unsigned char Percent)
{
    MTCX_DISPLAY_SERVICE_DATA data;
    int retvalue;
    unsigned char ErrorCode;

    if (Percent > 100)
        return -1;

    // Check if panel is linked.
    retvalue = MtcxSwitchToLinkedPanel(PanelNumber);
    if (retvalue != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    // Read display settings.
    retvalue = MtcxReadCommand(MTCX_CMD_DISPLAY_SERVICE, 0x00, PanelNumber, MTCX_DEV_PANEL,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    // Brightness not supported?
    if (data.Brightness == 255)
        return -1;

    // Write display settings.
    data.Brightness = Percent;
    retvalue = MtcxWriteCommand(MTCX_CMD_DISPLAY_SERVICE, 0x00, PanelNumber, MTCX_DEV_PANEL,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    return 0;
}
```

Listing 60: SetDisplayBrightness – Displayhelligkeit einstellen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.
Die aufgerufene Funktion **MtcxWriteCommand** finden Sie auf Seite 74.

7.12.2 Equalizer lesen/einstellen

Der Equalizer eines angeschlossenen Automation Panels kann über das **Panel Flags** Register des MTCX (siehe Seite 53) gelesen und eingestellt werden.

Der Equalizer ist im Automation Panel eingebaut und passt das DVI Signal für unterschiedliche Kabellängen an. Der Equalizerwert wird automatisch anhand der Kabellänge ermittelt. Sie können einen anderen Equalizerwert einstellen, um die beste optische Darstellung auf dem Display zu erreichen (z.B. bei schlechter Kabelqualität oder schlechter DVI Signalqualität).

Ein vom Anwender eingestellter Equalizerwert gilt nur bis zum nächsten Start des Systems. Wenn der Equalizerwert auch nach dem nächsten Systemstart verwendet werden soll, müssen Sie diesen in Ihrer Anwendung speichern und beim nächsten Systemstart nochmals einstellen.

Information:

Diese Funktion wird nur für Panel 0 bis 14 unterstützt. Die Displayeinheit eines PPC900 (Panel 15) und AP800 Extension Units unterstützen keinen Equalizer. Ob es sich um eine AP800 Extension Unit handelt, kann über das Panel Version Register (siehe Seite 51 und Codebeispiel auf Seite 92) ermittelt werden.

Das folgende C Codebeispiel zeigt, wie der Equalizerwert eines Displays gelesen werden kann.

```
// Get display equalizer value.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   EqualizerMode
//     [out] Points to a variable that receives the
//     equalizer mode: 0 = automatic or 1 = user specific.
//   EqualizerValue
//     [out] Points to a variable that receives the
//     equalizer value: 0 (strong) to 15 (weak).
//     If EqualizerMode is 0, this is the automatic
//     equalizer value. If EqualizerMode is 1, this
//     is the user specified equalizer value.
//
// Return 0 at success and -1 at failure.
//
// Remarks
//   AP900: Requires SDL firmware version 1.04 or higher.
int GetDisplayEqualizer(unsigned int PanelNumber, int *EqualizerMode, unsigned char
*EqualizerValue)
{
    int retvalue;

    // TODO: acquire synchronization object for panel data here

    retvalue = MtcxSwitchToLinkedPanel(PanelNumber);
    if (retvalue == MTCX_OKAY)
    {
        MTCX_PANEL_FLAGS_REG reg;

        *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_PANEL_FLAGS_ADDR);
        if (reg.EqualizerSupport)
        {
            *EqualizerMode = reg.EqualizerMode;
            if (reg.EqualizerMode)
                *EqualizerValue = reg.EqualizerUser;
            else
                *EqualizerValue = reg.EqualizerAuto;
            WRITE_PORT_ULONG(MTCX_PANEL_FLAGS_ADDR, *(unsigned long*)&reg);
        }
        else
            retvalue = MTCX_FAIL_NOSUPPORT;
    }

    // TODO: release synchronization object for panel data here

    if (retvalue != MTCX_OKAY)
    {
         // TODO: add your error handling here
         return -1;
    }

    return 0;
}
```

Listing 61: GetDisplayEqualizer – Display Equalizerwert lesen

Das folgende C Codebeispiel zeigt, wie der Equalizerwert eines Displays eingestellt werden kann.

```
// Set display equalizer value.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   EqualizerMode
//     [in] Specifies the equalizer mode:
//     0 = automatic or 1 = user specific.
//   EqualizerValue
//     [in] Specifies the equalizer value: 0 (strong) to 15 (weak).
//     If EqualizerMode is 1, this value overrides the
//     automatic equalizer value. If EqualizerMode is 0,
//     this value is ignored and the automatic equalizer
//     value is used.
//
// Return 0 at success and -1 at failure.
//
// Remarks
//   AP900: Requires firmware version 1.04.
int SetDisplayEqualizer(unsigned int PanelNumber, int EqualizerMode, unsigned char EqualizerVal-
ue)
{
    int retvalue;

    // TODO: acquire synchronization object for panel data here

    retvalue = MtcxSwitchToLinkedPanel(PanelNumber);
    if (retvalue == MTCX_OKAY)
    {
        MTCX_PANEL_FLAGS_REG reg;

        *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_PANEL_FLAGS_ADDR);
        if (reg.EqualizerSupport)
        {
            reg.EqualizerMode = EqualizerMode;
                   if (reg.EqualizerMode)
                reg.EqualizerUser = EqualizerValue;
            WRITE_PORT_ULONG(MTCX_PANEL_FLAGS_ADDR, *(unsigned long*)&reg);
        }
        else
            retvalue = MTCX_FAIL_NOSUPPORT;
    }

    // TODO: release synchronization object for panel data here

    if (retvalue != MTCX_OKAY)
    {
         // TODO: add your error handling here
         return -1;
    }

    return 0;
}
```

Listing 62: SetDisplayEqualizer – Display Equalizerwert einstellen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxSwitchToLinkedPanel** finden Sie auf Seite 68.

7.13 Tastenfunktionen

An einem APC910 und PPC900 können ein oder mehrere Automation Panels angeschlossen werden. Auf einem Automation Panel und der Displayeinheit eines PPC900 können jeweils bis zu 128 Matrixtasten vorhanden sein, die in 4 Tastenebenen bedient werden können und für die Sie folgende Funktionen ausführen können:

- Tastenanzahl lesen
- Tastenmatrix lesen
- Schlüsselschalter lesen
- Scancodes sperren und freigeben
- Status der Tastenkonfiguration lesen
- Modus der Tastenebene lesen
- Nummer der Tastenebene lesen und setzen

Die Tastenfunktionen werden vom MTCX ausgeführt.

Die folgenden Codebeispiele zeigen, wie Sie die Matrixtasten bedienen und auswerten können.

7.13.1 Tastenanzahl lesen

Die Anzahl der unterstützen Tasten eines Panels kann durch Lesen des **Panel Specials** Register des MTCX (siehe Seite 52) ermittelt werden.

Information:

Derzeit wird technisch bedingt von allen B&R Geräten, die Tasten unterstützen können, die Anzahl der Tasten mit 128 gemeldet. Davon kann aber nicht abgeleitet werden, dass auf dem Gerät wirklich 128 Tasten vorhanden sind. Es können auch weniger oder sogar keine Tasten vorhanden sein. Nur wenn Tastenanzahl 0 gelesen wird, kann davon ausgegangen werden, dass das Gerät keine Tasten unterstützt.

Das folgende C Codebeispiel zeigt, wie die Anzahl der unterstützten Tasten gelesen werden kann.

Aus Performancegründen wird nur das höherwertigste Byte des **Panel Specials** Register gelesen.

```
// Get supported key count of a panel.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   KeyCount
//     [out] Points to variable that receives the
//     maximum number of supported matrix keys: 0 to 128.
//
// Return 0 at success and -1 at failure.
int GetKeyCount(unsigned int PanelNumber, unsigned char *KeyCount)
{
    int retvalue;

    // TODO: acquire synchronization object for panel data here

    retvalue = MtcxSwitchToLinkedPanel(PanelNumber);
    if (retvalue == MTCX_OKAY)
        *KeyCount = READ_PORT_UCHAR(MTCX_PANEL_SPECIALS_ADDR + 2); // read byte 2 only

    // TODO: release synchronization object for panel data here

    if (retvalue != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    return 0;
}
```

Listing 63: GetKeyCount – Tastenanzahl lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxSwitchToLinkedPanel** finden Sie auf Seite 68.

7.13.2 Tastenmatrix lesen

Die Zustände der Matrixtasten eines Panels können über die **Panel Key Matrix** Register des MTCX (siehe Seite 54) gelesen werden.

Hinweis: Die Tastenzustände können auch gelesen werden, wenn keine gültige Tastenkonfiguration im System vorhanden ist.

Das folgende C Codebeispiel zeigt, wie die Tastenmatrix eines angeschlossenen Panels gelesen werden kann.

```
// Read key matrix of a panel.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   KeyMatrix
//     [out] Points to a buffer that receives the
//     key matrix: 1 bit per key; 0 = key released, 1 = key pressed.
//   Size
//     [in] Specifies the size of the buffer in bytes: 1 to 16.
//
// Return 0 at success and -1 at failure.
int GetKeyMatrix(unsigned int PanelNumber, unsigned char *KeyMatrix, int Size)
{
    int retvalue;

    if (Size < 1 || Size > 16)
        return -1;

    // TODO: acquire synchronization object for panel data here

    retvalue = MtcxSwitchToLinkedPanel(PanelNumber);
    if (retvalue == MTCX_OKAY)
    {
        int Port = MTCX_PANEL_KEY_MATRIX0_ADDR;

        while (Size-- > 0)
            *KeyMatrix++ = ~READ_PORT_UCHAR(Port++); // make pressed keys 1 instead of 0
    }

    // TODO: release synchronization object for panel data here

    if (retvalue != MTCX_OKAY)
    {
         // TODO: add your error handling here
         return -1;
    }

    return 0;
}
```

Listing 64: GetKeyMatrix – Tastenmatrix lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxSwitchToLinkedPanel** finden Sie auf Seite 68.

7.13.3 Schlüsselschalter lesen

Die Zustände der Schlüsselschalter eines Panels können durch Lesen des **Panel Specials** Register des MTCX (siehe Seite 52) gelesen werden.

Die Schalterzustände können nur gelesen werden, wenn eine gültige Tastenkonfiguration im System vorhanden ist. Eine Tastenkonfiguration kann mit dem B&R Key Editor erstellt und auf das Gerät geladen werden.

Information:

Die Tastenkonfiguration muss mit B&R Key Editor 2.50 oder höher erstellt werden.

Das folgende C Codebeispiel zeigt, wie die Zustände der Schlüsselschalter eines angeschlossenen Panels gelesen werden können.

```
// Read key switch states of a panel.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   KeySwitches
//     [out] Points to a variable that receives the
//     key switches: 00h to FFh.
//
// Return 0 at success and -1 at failure.
int GetKeySwitches(unsigned int PanelNumber, unsigned char *KeySwitches)
{
    int retvalue;

    // TODO: acquire synchronization object for panel data here

    retvalue = MtcxSwitchToLinkedPanel(PanelNumber);
    if (retvalue == MTCX_OKAY)
        // make closed contacts 1 instead of 0
        *KeySwitches = ~READ_PORT_UCHAR(MTCX_PANEL_SPECIALS_ADDR + 1); // read byte 1 only

    // TODO: release synchronization object for panel data here

    if (retvalue != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    return 0;
}
```

Listing 65: GetKeySwitches – Schlüsselschalter lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxSwitchToLinkedPanel** finden Sie auf Seite 68.

7.13.4 Scancodesperre der Matrixtasten lesen/setzen

Die Scancodesperre der Matrixtasten eines Panels können mit dem **Key Service** Kommando des MTCX (siehe Seite 22) gelesen und gesetzt werden.

Diese Funktion kann zum Sperren der Scancodes verwendet werden, wenn z.B. für Testzwecke die Tastenmatrix gelesen wird und das Betriebssystem währenddessen nicht auf Tastendrücke reagieren soll.

Das folgende C Codebeispiel zeigt, wie ermittelt werden kann, ob die Scancodes der Matrixtasten eines Panels gesperrt oder freigegeben sind.

Ob die Scancodes eines Panels gesperrt sind, kann auch schneller über das **Panel Flags** Register des MTCX (siehe Seite 53) gelesen werden (siehe **MtcxAreScanCodesLocked** Codebeispiel auf Seite 79).

```
// Get scan code lock state of a panel.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   Locked
//     [out] Points to a variable that receives the
//     lock state: 1 = locked, 0 = unlocked.
//
// Return 0 at success and -1 at failure.
int GetScanCodeLock(unsigned int PanelNumber, int *Locked)
{
    MTCX_SCANCODE_LOCK_DATA data;
    unsigned char ErrorCode;
    int retvalue;

    // Read lock bits.
    retvalue = MtcxReadCommand(MTCX_CMD_KEY_SERVICE, 0, 0, MTCX_DEV_BASEBOARD,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    *Locked = (data.LockBits & (1<<PanelNumber)) ? 1 : 0;
    return 0;
}
```

Listing 66: GetScanCodeLock – Scancodesperre lesen

Das folgende C Codebeispiel zeigt, wie die Scancodes der Matrixtasten eines Panels gesperrt und freigegeben werden können.

Wenn Lock gleich 0 ist, werden die Scancodes freigegeben und die zugeordneten Tastencodes werden an das Betriebssystem gesendet, wenn eine Taste gedrückt wird. Wenn Lock ungleich 0 ist, sind die Scancodes gesperrt und das Betriebssystem empfängt keine Tastencodes. Es können dann nur die Zustände der Tasten gelesen werden.

```
// Set scan code lock state of a panel.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   Lock
//     [in] Specifies the lock state: 1 = locked, 0 = unlocked.
//
// Return 0 at success and -1 at failure.
int SetScanCodeLock(unsigned int PanelNumber, int Lock)
{
    MTCX_SCANCODE_LOCK_DATA data;
    unsigned char ErrorCode;
    int retvalue;

    // Read, modify, write scan code lock bits.
    retvalue = MtcxReadCommand(MTCX_CMD_KEY_SERVICE, 0, 0, MTCX_DEV_BASEBOARD,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }
    if (Lock)
        data.LockBits |= (1<<PanelNumber);
    else
        data.LockBits &= ~(1<<PanelNumber);
    retvalue = MtcxWriteCommand(MTCX_CMD_KEY_SERVICE, 0, 0, MTCX_DEV_BASEBOARD,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    return 0;
}
```

Listing 67: SetScanCodeLock – Scancodesperre setzen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.
Die aufgerufene Funktion **MtcxWriteCommand** finden Sie auf Seite 74.

7.13.5 Status der Tastenkonfiguration lesen

Der Status der Tastenkonfiguration im Flashspeicher (gültig/ungültig) kann durch Lesen des **Baseboard KeyState** Register des MTCX (siehe Seite 44) ermittelt werden. Eine Tastenkonfiguration kann mit dem B&R Key Editor erstellt und auf das Gerät geladen werden.

Das folgende C Codebeispiel zeigt, wie der Status der Tastenkonfiguration gelesen werden kann.

```
// Read the state of the key configuration.
//
// Parameters
//   Valid
//     [out] Points to a variable that receives the
//     state of the key configuration:
//     1 indicates the data is valid,
//     0 indicates the data is not valid (for example has a
//       wrong checksum or no data is programmed into the Flash memory).
void GetKeyCfgState(int *Valid)
{
    MTCX_BASEBOARD_KEYSTATE_REG reg;

    *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_BASEBOARD_KEYSTATE_ADDR);
    *Valid = reg.Status ? 1 : 0;
}
```

Listing 68: GetKeyCfgState – Status der Tastenkonfiguration lesen

Die verwendeten Definitionen finden Sie auf Seite 59.

7.13.6 Modus der Tastenebene lesen

Der Modus der Tastenebene kann über das **Baseboard KeyState** Register des MTCX (siehe Seite 44) ermittelt werden.

Die Tasten eines B&R Geräts können in vier verschiedenen Ebenen bedient werden, d.h. eine Taste kann bis zu vier verschiedene Tastencodes senden. Die Umschaltung der Ebene kann mit speziellen Ebenentasten erfolgen. Diese Ebenentasten und der Modus zur Umschaltung der Tastenebene werden mit dem B&R Key Editor konfiguriert.

Das folgende C Codebeispiel zeigt, wie der Modus der Tastenebene gelesen werden kann.

Folgende Modi sind möglich:
0 = Shift Mode: Die Tastenebene wird nur solange umgeschaltet, während eine Ebenentaste gedrückt ist.
1 = Toggle Mode: Die Tastenebene wird bei jedem Drücken einer Ebenentaste umgeschaltet.
2 = One-Shot Mode: Die Tastenebene bleibt nach dem Drücken einer Ebenentaste nur bis zum ersten Drücken einer Taste umgeschaltet

```
// Get key layer mode.
//
// Parameters
//   KeyLayerMode
//     [out] Points to a variable that receives the
//     key layer mode: 0 = shift mode, 1 = toggle mode,
//     2 = one-shot mode.
void GetKeyLayerMode(unsigned char *KeyLayerMode)
{
    MTCX_BASEBOARD_KEYSTATE_REG reg;

    *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_BASEBOARD_KEYSTATE_ADDR);
    *KeyLayerMode = reg.Mode;
}
```

Listing 69: GetKeyLayerMode – Modus der Tastenebene lesen

Die verwendeten Definitionen finden Sie auf Seite 59.

7.13.7 Tastenebene lesen/einstellen

Die aktuelle Tastenebene kann über das **Baseboard KeyState** Register des MTCX (siehe Seite 44) gelesen und eingestellt werden.

Die Tasten eines B&R Geräts können in vier verschiedenen Ebenen bedient werden, d.h. eine Taste kann bis zu vier verschiedene Tastencodes senden. Die Umschaltung der Ebene kann mit speziellen Ebenentasten erfolgen. Diese werden mit dem B&R Key Editor konfiguriert.

Das folgende C Codebeispiel zeigt, wie die aktuelle Tastenebene gelesen werden kann.

```
// Get key layer.
//
// Parameters
//   KeyLayer
//     [out] Points to a variable that receives the
//     key layer: 0 to 3.
void GetKeyLayer(unsigned char *KeyLayer)
{
    MTCX_BASEBOARD_KEYSTATE_REG reg;

    *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_BASEBOARD_KEYSTATE_ADDR);
    *KeyLayer = reg.Layer;
}
```

Listing 70: GetKeyLayer – Tastenebene lesen

Das folgende C Codebeispiel zeigt, wie die Tastenebene eingestellt werden kann.

Information:

Die Tastenebene wird nur eingestellt, wenn eine gültige Tastenkonfiguration auf dem Gerät vorhanden ist (siehe Seite 133) - das Kommando liefert aber keinen Fehler!

```
// Set key layer.
//
// Parameters
//   KeyLayer
//     [in] Specifies the key layer: 0 to 3.
//
// Return 0 at success and -1 at failure.
//
// Remarks
//   Requires a valid key configuration!
int SetKeyLayer(unsigned char KeyLayer)
{
    MTCX_KEY_LAYER_DATA data;
    unsigned char ErrorCode;
    int retvalue;

    if (KeyLayer > 3)
        return -1;

    memset(&data, 0x00, sizeof(data));
    data.Layer = KeyLayer;

    retvalue = MtcxWriteCommand(MTCX_CMD_KEY_LAYER, 0, 0, MTCX_DEV_BASEBOARD,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    return 0;
}
```

Listing 71: SetKeyLayer – Tastenebene einstellen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxWriteCommand** finden Sie auf Seite 74.

7.14 LED Funktionen

An einem APC910 und PPC900 können ein oder mehrere Automation Panels angeschlossen werden. Auf einem Automation Panel und der Displayeinheit eines PPC900 können analog zu den Matrixtasten auch jeweils bis zu 128 LEDs vorhanden sein, die in 4 Ebenen angesteuert werden können und für die Sie folgende Funktionen ausführen können:

- LED Anzahl lesen
- LED Matrix lesen und setzen
- Einzelne LED lesen und setzen
- Run LED lesen und setzen

Die LED Funktionen werden vom MTCX ausgeführt.

Die folgenden Codebeispiele zeigen, wie Sie die Matrix-LEDs bedienen und auswerten können. Dazu werden die folgenden Datentypen benötigt:

```
typedef unsigned char MTCX_LAYER_LED_DATA[16]; // 64 LED states, 2 bits for each LED

typedef struct
{
    unsigned short Offset   :8; // offset of first LED to read: 0 to 63
    unsigned short Size     :6; // number of LEDs to read - 1
    unsigned short Reserve  :2;
} MTCX_LAYER_LED_ADDR;
```

Listing 72: Datentypen für LED Funktionen

7.14.1 LED Anzahl lesen

Die Anzahl der unterstützten LEDs eines Panels kann durch Lesen des **Panel Specials** Register des MTCX (siehe Seite 52) ermittelt werden.

Information:

Derzeit wird technisch bedingt von allen B&R Geräten, die LEDs unterstützen können, die Anzahl der LEDs mit 128 gemeldet. Davon kann aber nicht abgeleitet werden, dass auf dem Gerät wirklich 128 LEDs vorhanden sind. Es können auch weniger oder sogar keine LEDs vorhanden sein. Nur wenn LED Anzahl 0 gelesen wird, kann davon ausgegangen werden, dass das Gerät keine LEDs unterstützt.

Das folgende C Codebeispiel zeigt, wie die Anzahl der unterstützten LEDs gelesen werden kann.

Aus Performancegründen wird nur das benötigte Byte des **Panel Specials** Register gelesen.

```
// Get available LED count.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   LedCount
//     [out] Points to variable that receives the
//     maximum number of supported matrix LEDs: 0 to 128.
//
// Return 0 at success and -1 at failure.
int GetLedCount(unsigned int PanelNumber, unsigned char *LedCount)
{
    int retvalue;

    // TODO: acquire synchronization object for panel data here

    retvalue = MtcxSwitchToLinkedPanel(PanelNumber);
    if (retvalue == MTCX_OKAY)
        *LedCount = READ_PORT_UCHAR(MTCX_PANEL_SPECIALS_ADDR + 3); // read byte 3 only

    // TODO: release synchronization object for panel data here

    if (retvalue != MTCX_OKAY)
    {
         // TODO: add your error handling here
         return -1;
    }

    return 0;
}
```

Listing 73: GetLedCount – LED Anzahl lesen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxSwitchToLinkedPanel** finden Sie auf Seite 68.

7.14.2 LED Matrix lesen/setzen

Die Zustände der Matrix-LEDs eines Panels können mit dem **LED Service** Kommando des MTCX (siehe Seite 26) gelesen und gesetzt werden.

LEDs können wie die Tasten in vier verschiedenen Ebenen bedient werden.

Das folgende C Codebeispiel zeigt, wie die gesamte LED Matrix eines angeschlossenen Panels gelesen werden kann.

Jede LED wird durch zwei Bits dargestellt: 00b = LED aus, 01b = LED blinkt langsam, 10b = LED blinkt schnell, 11b = LED ein.

```
// Read LED matrix.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   Layer
//     [in] Specifies the layer: 0 to 3.
//   LedMatrix
//     [in] Points to a buffer that receives the
//     states of 128 LED (2 bits per LED).
//   Size
//     [in] Specifies the size of the buffer in bytes: 32.
//
// Return 0 at success and -1 at failure.
int GetLedMatrix(unsigned int PanelNumber, unsigned char Layer, unsigned char *LedMatrix, int
Size)
{
    MTCX_LAYER_LED_ADDR LedAddr;
    unsigned char ErrorCode;
    int retvalue;

    if (Layer > 3)
        return -1;

    if (Size != 32)
        return -1;

    LedAddr.Reserve = 0;
    LedAddr.Offset = 0;
    LedAddr.Size = 63;

    // Check if panel is linked.
    retvalue = MtcxSwitchToLinkedPanel(PanelNumber);
    if (retvalue != MTCX_OKAY)
    {
         // TODO: add your error handling here
         return -1;
    }

    // Read LEDs 0 to 63.
    if ((retvalue = MtcxReadCommand(MTCX_CMD_LED_SERVICE0 + Layer, *(unsigned short*)&LedAddr,
PanelNumber,
        MTCX_DEV_PANEL, LedMatrix, 16, &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    LedMatrix += 16;
    LedAddr.Offset = 64;

    // Read LEDs 64 to 127.
    if ((retvalue = MtcxReadCommand(MTCX_CMD_LED_SERVICE0 + Layer, *(unsigned short*)&LedAddr,
PanelNumber,
```

```
        MTCX_DEV_PANEL, LedMatrix, 16, &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    return 0;
}
```

Listing 74: GetLedMatrix – LED Matrix lesen

Das folgende C Codebeispiel zeigt, wie die gesamte LED Matrix eines angeschlossenen Panels geschrieben werden kann.

```
// Write LED matrix.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   Layer
//     [in] Specifies the layer: 0 to 3.
//   LedMatrix
//     [in] Points to a buffer that contains the
//     states of 128 LED (2 bits per LED).
//   Size
//     [in] Specifies the size of the buffer in bytes: 32.
//
// Return 0 at success and -1 at failure.
int SetLedMatrix(unsigned int PanelNumber, unsigned char Layer, unsigned char *LedMatrix, int
Size)
{
    MTCX_LAYER_LED_ADDR LedAddr;
    unsigned char ErrorCode;
    int retvalue;

    if (Layer > 3)
        return -1;

    if (Size != 32)
        return -1;

    // Check if panel is linked.
    retvalue = MtcxSwitchToLinkedPanel(PanelNumber);
    if (retvalue != MTCX_OKAY)
    {
         // TODO: add your error handling here
         return -1;
    }

    LedAddr.Reserve = 0;
    LedAddr.Offset = 0;
    LedAddr.Size = 63;

    // Write LEDs 0 to 63.
    if ((retvalue = MtcxWriteCommand(MTCX_CMD_LED_SERVICE0 + Layer, *(unsigned short*)&LedAddr,
PanelNumber,
        MTCX_DEV_PANEL, LedMatrix, 16, &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    LedMatrix += 16;
    LedAddr.Offset = 64;

    // Write LEDs 64 to 127.
    if ((retvalue = MtcxWriteCommand(MTCX_CMD_LED_SERVICE0 + Layer, *(unsigned short*)&LedAddr,
PanelNumber,
        MTCX_DEV_PANEL, LedMatrix, 16, &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
```

```
            return -1;
        }

        return 0;
    }
```

Listing 75: SetLedMatrix – LED Matrix schreiben

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxSwitchToLinkedPanel** finden Sie auf Seite 68.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.
Die aufgerufene Funktion **MtcxWriteCommand** finden Sie auf Seite 74.

7.14.3 Einzelne LED lesen/setzen

Der Zustand einer einzelnen LED eines Panels kann ebenfalls mit dem **LED Service** Kommando des MTCX (siehe Seite 26) gelesen und gesetzt werden.

LEDs können wie die Tasten in vier verschiedenen Ebenen bedient werden.

Das folgende C Codebeispiel zeigt, wie eine einzelne LED eines angeschlossenen Panels gelesen werden kann.

Mögliche LED Zustände (LedState): 0 = LED aus, 1 = LED blinkt langsam, 2 = LED blinkt schnell, 3 = LED ein

```
// Get the state of a LED.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   Layer
//     [in] Specifies the layer: 0 to 3.
//   LedNumber
//     [in] Specifies the LED number: 0 to 127.
//   LedState
//     [ount] Points to a variable that receives the
//     LED state: 0 = off, 1 = slow blinking,
//     2 = fast blinking, 3 = on.
//
// Return 0 at success and -1 at failure.
int GetLed(unsigned int PanelNumber, unsigned char Layer, unsigned char LedNumber, unsigned char
*LedState)
{
    MTCX_LAYER_LED_ADDR LedAddr;
    MTCX_LAYER_LED_DATA LedData;
    unsigned char ErrorCode;
    int retvalue;

    if (Layer > 3)
        return -1;
    if (LedNumber >= 128)
        return -1;

    // Check if panel is linked.
    retvalue = MtcxSwitchToLinkedPanel(PanelNumber);
    if (retvalue != MTCX_OKAY)
    {
         // TODO: add your error handling here
         return -1;
    }

    LedAddr.Reserve = 0;
    LedAddr.Offset = LedNumber;
    LedAddr.Size = 0; // 0 = one LED

    if ((retvalue = MtcxReadCommand(MTCX_CMD_LED_SERVICE0 + Layer, *(unsigned short*)&LedAddr,
PanelNumber,
        MTCX_DEV_PANEL, &LedData, sizeof(LedData), &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    *LedState = LedData[0] & 0x3;
    return 0;
}
```

Listing 76: GetLed – einzelne LED lesen

Das folgende C Codebeispiel zeigt, wie eine einzelne LED eines angeschlossenen Panels gesetzt werden kann.

```
// Set the state of a LED.
//
// Parameters
//   PanelNumber
//     [in] Specifies the panel number: 0 to 15.
//   Layer
//     [in] Specifies the layer: 0 to 3.
//   LedNumber
//     [in] Specifies the LED number: 0 to 127.
//   LedState
//     [in] Specifies the LED state: 0 = off, 1 = slow blinking,
//     2 = fast blinking, 3 = on.
//
// Return 0 at success and -1 at failure.
int SetLed(unsigned int PanelNumber, unsigned char Layer, unsigned char LedNumber, unsigned char
LedState)
{
    MTCX_LAYER_LED_ADDR LedAddr;
    MTCX_LAYER_LED_DATA LedData;
    unsigned char ErrorCode;
    int retvalue;

    if (Layer > 3)
        return -1;
    if (LedNumber >= 128)
        return -1;
    if (LedState > 3)
        return -1;

    // Check if panel is linked.
    retvalue = MtcxSwitchToLinkedPanel(PanelNumber);
    if (retvalue != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    LedAddr.Reserve = 0;
    LedAddr.Offset = LedNumber;
    LedAddr.Size = 0; // 0 = one LED
    LedData[0] = LedState;

    if ((retvalue = MtcxWriteCommand(MTCX_CMD_LED_SERVICE0 + Layer, *(unsigned short*)&LedAddr,
PanelNumber,
        MTCX_DEV_PANEL, &LedData, sizeof(LedData), &ErrorCode)) != MTCX_OKAY)
    {
        // TODO: add your error handling here
        return -1;
    }

    return 0;
}
```

Listing 77: SetLed – einzelne LED setzen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxSwitchToLinkedPanel** finden Sie auf Seite 68.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.
Die aufgerufene Funktion **MtcxWriteCommand** finden Sie auf Seite 74.

7.14.4 Run LED lesen/setzen

Die Run LED kann über das **Config 2nd** Register des MTCX (siehe Seite 13) gelesen und eingestellt werden.

Das folgende C Codebeispiel zeigt, wie die Run LED gelesen werden kann.

```
// Read Run LED state.
//
// Parameters
//   State
//     [out] Points to a variable that receives the
//     LED state: 0 = off, 1 = on.
void GetRunLed(int *State)
{
    MTCX_CONFIG_2ND_REG reg;

    *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_CONFIG_2ND_ADDR);

    *State = reg.RunLedState;
}
```

Listing 78: GetRunLed – Zustand der Run LED lesen

Das folgende C Codebeispiel zeigt, wie die Run LED gesetzt werden kann.

```
// Set Run LED state.
//
// Parameters
//   State
//     [in] Specifies the LED state: 0 = off, 1 = on.
void SetRunLed(int State)
{
    MTCX_CONFIG_2ND_REG reg;

    *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_CONFIG_2ND_ADDR);

    reg.RunLedState = State;
    WRITE_PORT_ULONG(MTCX_CONFIG_2ND_ADDR, *(unsigned long*)&reg);
}
```

Listing 79: SetRunLed – Zustand der Run LED setzen

Die verwendeten Definitionen finden Sie auf Seite 59.

7.15 Watchdogbedienung, Software Reset

Der Watchdog ist eine Lebensüberwachung, die von einer PC Anwendung jederzeit ein- und ausgeschaltet werden kann.

Die Watchdog Überwachung wird vom MTCX durchgeführt.

Der Watchdog muss von der PC Anwendung über die MTCX Kommandoschnittstelle konfiguriert werden. Bei der Konfiguration wird ein Zeitfenster für das Quittieren des Watchdog eingestellt.

Das Quittieren des Watchdog erfolgt durch Beschreiben des Watchdog Toggle Bit. Nach der Konfiguration kann der Watchdog von der PC Anwendung durch das erstmalige Beschreiben des Toggle Bit aktiviert werden. Ab diesem Zeitpunkt muss die PC Anwendung den Watchdog innerhalb des Zeitfensters quittieren.

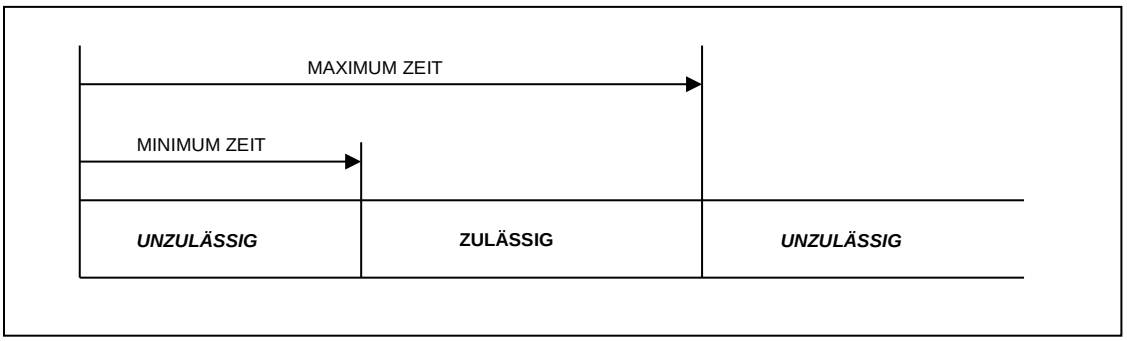

Abbildung 2: Watchdog Zeiten

Tritt nun eine Zeitverletzung ein (z.B. wegen Programmabsturz oder Dauerschleife der PC Anwendung), wird vom MTCX die NMI Logik des PC aktiviert. Wenn die NMI Logik ausgeschaltet ist, wird sofort ein Reset eingeleitet. Nach jedem Reset ist der Watchdog wieder ausgeschaltet.

Über das Wachdog Service Kommando können Sie auch gezielt per Software einen Reset des PC durchführen.

Vorsicht!

Nicht gesicherte Daten des PC gehen durch den Reset verloren!

Die folgenden Codebeispiele zeigen, wie Sie den Watchdog bedienen oder selbst einen Reset des PC ausführen können.

7.15.1 Watchdog Zeitfenster einstellen

Damit der Watchdog verwendet werden kann, muss als erstes mit dem **Watchdog Service** Kommando des MTCX (siehe Seite 28) das Zeitfenster für das Quittieren (Toggeln) des Watchdog eingestellt werden.

Das Zeitfenster wird über eine Minimal- und Maximalzeit in Millisekunden angegeben. Im Normalfall wird die Minimalzeit mit 0 Millisekunden eingestellt.

Information:

Stellen Sie eine möglichst große Maximalzeit ein, um die Systembelastung durch das Quittieren des Watchdog gering zu halten.

Das folgende C Codebeispiel zeigt, wie das Zeitfenster des Watchdog gelesen werden kann.

```
// Read minimum and maximum watchdog time.
//
// Parameters
//   MinTime
//     [out] Points to a variable that receives the
//     minimum watchdog time in milliseconds: 0 to 65535.
//   MaxTime
//     [out] Points to a variable that receives the
//     maximum watchdog time in milliseconds: 0 to 65535.
//
// Return 0 at success and -1 at failure.
int GetWatchdogTime(unsigned short *MinTime, unsigned short *MaxTime)
{
    MTCX_WATCHDOG_CONFIG_DATA data;
    int retvalue;
    unsigned char ErrorCode;

    retvalue = MtcxReadCommand(MTCX_CMD_WDOG_SERVICE, 0, 0, MTCX_DEV_BASEBOARD,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
         // TODO: add your error handling here
         return -1;
    }

    *MinTime = data.MinTime;
    *MaxTime = data.MaxTime;
    return 0;
}
```

Listing 80: GetWatchdogTime – Watchdog Zeitfenster lesen

Das folgende C Codebeispiel zeigt, wie das Zeitfenster des Watchdog eingestellt werden kann.

```
// Set minimum and maximum watchdog time.
//
// Parameters
//   MinTime
//     [in] Specifies the minimum watchdog time
//     in milliseconds: 0 to 65535.
//   MaxTime
//     [in] Specifies the maximum watchdog time
//     in milliseconds: 0 or 65535.
//
// Return 0 at success and -1 at failure.
//
// Remarks
//   Set MinTime and MaxTime to 0 to deactivate
//   the watchdog.
int SetWatchdogTime(unsigned short MinTime, unsigned short MaxTime)
{
    MTCX_WATCHDOG_CONFIG_DATA data;
    int retvalue;
    unsigned char ErrorCode;

    data.MinTime = MinTime;
    data.MaxTime = MaxTime;

    retvalue = MtcxWriteCommand(MTCX_CMD_WDOG_SERVICE, 0, 0, MTCX_DEV_BASEBOARD,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
         // TODO: add your error handling here
         return -1;
    }
    return 0;
}
```

Listing 81: SetWatchdogTime – Watchdog Zeitfenster schreiben

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.
Die aufgerufene Funktion **MtcxWriteCommand** finden Sie auf Seite 74.

7.15.2 Watchdog toggeln

Nachdem das Zeitfenster des Watchdog eingestellt wurde, kann der Watchdog durch erstmaliges Beschreiben des Toggle Bit im Baseboard Support Register (siehe Seite 43) aktiviert werden. Anschließend muss das Toggle Bit innerhalb des konfigurierten Zeitfensters beschrieben werden, bis der Watchdog durch Einstellen eines neuen Zeitfensters wieder deaktiviert wird.

Das Toggle Bit muss von der PC Anwendung immer auf 0 gesetzt werden. Das Toggle Bit wird vom MTCX jede Millisekunde auf 1 gesetzt.

Vorsicht!

Wenn das Toggle Bit nicht innerhalb des konfigurierten Zeitfensters auf 0 gesetzt wird, erfolgt ein Neustart des PCs. Dabei kann es zu Datenverlusten kommen!

Das folgende C Codebeispiel zeigt, wie das Watchdog Toggle Bit in C beschrieben werden kann.

```
// Toggle watchdog.
void ToggleWatchdog()
{
    MTCX_BASEBOARD_SUPPORT_REG reg;

    *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_BASEBOARD_SUPPORT_ADDR);
    reg.WatchdogToggle = 0;
    WRITE_PORT_ULONG(MTCX_BASEBOARD_SUPPORT_ADDR, *(unsigned long*)&reg);
}
```

Listing 82: ToggleWatchdog – Watchdog toggeln

Die verwendeten Definitionen finden Sie auf Seite 59.

7.15.3 Software Reset

Über das **Watchdog Service** Kommando des MTCX (siehe Seite 28) können Sie per Software einen Reset des PC durchführen.

Das folgende C Codebeispiel zeigt, wie der Reset durchgeführt werden kann.

```
// Generate a software reset.
// No return at success and -1 at failure.
int SoftwareReset()
{
    int retvalue;
    unsigned char ErrorCode;

    retvalue = MtcxWriteCommand(MTCX_CMD_WDOG_SERVICE, 0x00, 0x02, MTCX_DEV_BASEBOARD,
        0, 0, &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
         // TODO: add your error handling here
         return -1;
    }
    // We should not reach this point...
    return -1;
}
```

Listing 83: SoftwareReset – Reset des PC durchführen

Die Funktion kehrt bei erfolgreicher Ausführung nicht zurück.

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxWriteCommand** finden Sie auf Seite 74.

7.16 Anwenderfunktionen

Auf einem APC910 und PPC900 wird die Vergabe einer User Serial ID unterstützt. Die User Serial ID ist ein 32 Bit großer, frei definierbarer Wert. Mit der User Serial ID können Sie das verwendete B&R Gerät mit Ihrer eigenen Erkennungsnummer versehen. Die User Serial ID wird dauerhaft am B&R Gerät gespeichert.

Der Zugriff auf die User Serial ID wird über den MTCX durchgeführt.

Die folgenden Codebeispiele zeigen, wie Sie die User Serial ID lesen und einstellen können.

7.16.1 User Serial ID lesen/einstellen

Die User Serial ID kann mit dem **Device Info** Kommando des MTCX (siehe Seite 19) gelesen und eingestellt werden.

Das folgende C Codebeispiel zeigt, wie die User Serial ID gelesen werden kann.

```
// Read User Serial ID.
//
// Parameters
//   Id
//     [out] Points to a variable that receives the
//     User Serial ID: 00000000h to FFFFFFFFh.
//
// Return 0 at success and -1 at failure.
int GetUserSerialId(unsigned long *Id)
{
    MTCX_ID_SERVICE_DATA data;
    int retvalue;
    unsigned char ErrorCode;

    retvalue = MtcxReadCommand(MTCX_CMD_DEVICE_INFO, 0, 0, MTCX_DEV_BASEBOARD,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
         // TODO: add your error handling here
         return -1;
    }

    *Id = data.User;
    return 0;
}
```

Listing 84: GetUserSerialID – User Serial ID lesen

Das folgende C Codebeispiel zeigt, wie die User Serial ID eingestellt werden kann.

```
// Set User Serial ID.
//
// Parameters
//   Id
//     [in] Specifies the User Serial ID: 00000000h to FFFFFFFFh.
//
// Return 0 at success and -1 at failure.
int SetUserSerialId(unsigned long Id)
{
    MTCX_ID_SERVICE_DATA data;
    int retvalue;
    unsigned char ErrorCode;

    data.Reserve = 0;
    data.User = Id;

    retvalue = MtcxWriteCommand(MTCX_CMD_DEVICE_INFO, 0, 0, MTCX_DEV_BASEBOARD,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
         // TODO: add your error handling here
         return -1;
    }
    return 0;
}
```

Listing 85: SetUserSerialID – User Serial ID einstellen

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.
Die aufgerufene Funktion **MtcxWriteCommand** finden Sie auf Seite 74.

7.17 USV Funktionen

Für eine eingebaute USV IF Option können folgende Funktionen durchgeführt werden:

- USV „erkannt“ und „verbunden“ prüfen
- USV Status lesen
- USV Batteriespannung, -strom und -temperatur lesen
- USV Anwendereinstellungen lesen/schreiben
- USV abschalten

Der USV Funktionen werden vom MTCX ausgeführt.

Die folgenden Codebeispiele zeigen, wie Sie den Zustand und die Betriebsdaten der USV lesen, die Anwendereinstellungen der USV lesen und schreiben und die USV abschalten können.

7.17.1 USV „erkannt“ prüfen

Ob eine USV erkannt wurde (d.h. eingebaut ist), kann über das **UPS Status** Register des MTCX (siehe Seite 47) geprüft werden.

Das folgende C Codebeispiel zeigt, wie geprüft werden kann, ob eine USV „erkannt“ wurde.

```
// Check if UPS is detected.
//
// Return 1 if UPS is detected and 0 if not.
int IsUpsDetected()
{
    MTCX_UPS_STATUS_REG_BYTE0 reg;

    *(unsigned char*)&reg = READ_PORT_UCHAR(MTCX_UPS_STATUS_ADDR); // read byte 0 only
    return reg.Detected;
}
```

Listing 86: IsUpsDetected – USV “erkannt” prüfen

Die verwendeten Definitionen finden Sie auf Seite 59.

7.17.2 USV „verbunden“ prüfen

Ob eine USV verbunden ist (d.h. ob die Kommunikation mit der USV funktioniert), kann über das **UPS Status** Register des MTCX (siehe Seite 47) geprüft werden.

Das folgende C Codebeispiel zeigt, wie geprüft werden kann, ob eine USV verbunden ist.

```
// Check if UPS is linked
// (that means communication with the UPS is OK).
//
// Return 1 if UPS is linked and 0 if not.
int IsUpsLinked()
{
    MTCX_UPS_STATUS_REG_BYTE0 reg;

    *(unsigned char*)&reg = READ_PORT_UCHAR(MTCX_UPS_STATUS_ADDR); // read byte 0 only
    return reg.Linked;
}
```

Listing 87: IsUpsLinked – USV “angeschlossen” prüfen

Die verwendeten Definitionen finden Sie auf Seite 59.

7.17.3 USV Statusflags lesen

Der aktuelle Status der USV (z.B. ob Batteriebetrieb aktiv ist) kann über das **UPS Status** Register des MTCX (siehe Seite 47) gelesen werden.

Information:

Diese Funktion liefert nur gültige Werte, wenn die USV „verbunden“ ist (siehe Seite 151).

Das folgende C Codebeispiel zeigt, wie der USV Status gelesen werden kann. Der Status wird in Form einzelner Flags (siehe MTCX_UPS_STATUS_x Konstanten) geliefert.

Information:

Einige Statusbits werden vom MTCX nicht unterstützt (siehe Seite 47).

```
// Read UPS status flags.
//
// Parameters
//   Status
//     [out] Points to variable that receives the
//     UPS status flags.
void GetUpsStatusFlags(unsigned short *Flags)
{
    MTCX_UPS_STATUS_REG reg;

    *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_UPS_STATUS_ADDR);
    *Flags = reg.Flags;
}
```

Listing 88: GetUpsStatusFlags – USV Statusflags lesen

Die verwendeten Definitionen finden Sie auf Seite 59.

7.17.4 USV Batteriespannung lesen

Die aktuelle Spannung der USV Batterie kann über das **UPS Values** Register des MTCX (siehe Seite 48) gelesen werden.

Information:

Diese Funktion liefert nur gültige Werte, wenn die USV „verbunden“ ist (siehe Seite 151).

Das folgende C Codebeispiel zeigt, wie die USV Batteriespannung in mV gelesen werden kann.

```
// Read UPS battery voltage.
//
// Parameters
//   Voltage
//     [out] Points to variable that receives the
//     UPS battery voltage in mV.
void GetUpsBatVoltage(unsigned short *Voltage)
{
    MTCX_UPS_VALUES_REG2 reg;

    *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_UPS_VALUES_ADDR);
    *Voltage = reg.BatVoltage;
}
```

Listing 89: GetUpsBatVoltage – USV Batteriespannung lesen

Die verwendeten Definitionen finden Sie auf Seite 59.

7.17.5 USV Batteriestrom lesen

Der aktuelle Ladestrom der USV Batterie bzw. der Entladestrom (wenn die USV im Batteriebetrieb ist) kann über das **UPS Values** Register des MTCX (siehe Seite 48) gelesen werden.

Information:

Diese Funktion liefert nur gültige Werte, wenn die USV „verbunden“ ist (siehe Seite 151).

Das folgende C Codebeispiel zeigt, wie der USV Batteriestrom in mA gelesen werden kann.

```
// Read UPS battery current.
//
// Parameters
//   Current
//     [out] Points to variable that receives the
//     UPS battery current in mA.
//     Positive values indicate the charge current.
//     Negative values indicate the discharge current.
void GetUpsBatCurrent(short *Current)
{
    MTCX_UPS_VALUES_REG2 reg;

    *(unsigned long*)&reg = READ_PORT_ULONG(MTCX_UPS_VALUES_ADDR);
    *Current = reg.BatCurrent;
}
```

Listing 90: GetUpsBatCurrent – USV Batteriestrom lesen

Die verwendeten Definitionen finden Sie auf Seite 59.

7.17.6 USV Batterietemperatur lesen

Die aktuelle Temperatur der USV Batterie kann über das **UPS Status** Register des MTCX (siehe Seite 47) gelesen werden.

Information:

Diese Funktion liefert nur gültige Werte, wenn die USV „verbunden“ ist (siehe Seite 151).

Das folgende C Codebeispiel zeigt, wie die USV Batterietemperatur in Grad Celsius gelesen werden kann.

Hinweis: Wenn die Temperatur nicht gelesen werden kann, wird der Wert -128 geliefert,

```
// Read UPS battery temperature.
//
// Parameters
//   Temp
//     [out] Points to variable that receives the
//     UPS battery temperature in degrees Celsius.
//     -128 indicates the temperature cannot be read.
void GetUpsBatTemp(char *Temp)
{
    *Temp = (char)READ_PORT_UCHAR(MTCX_UPS_STATUS_ADDR + 3); // read byte 3 only
}
```

Listing 91: GetUpsBatTemperature – USV Batterietemperatur lesen

Die verwendeten Definitionen finden Sie auf Seite 59.

7.17.7 USV Anwendereinstellungen lesen/schreiben

Die Anwendereinstellungen der USV können mit dem **Module Correction** Kommando des MTCX (siehe Seite 34) gelesen und geschrieben werden.

Vorsicht!

Ein Low Battery Shutdown unterbricht einen eventuell bereits laufenden USV Shutdown (siehe Seite 157) und kann damit die geplante Abschaltzeit des Systems verkürzen!

Das folgende C Codebeispiel zeigt, wie die USV Anwendereinstellungen gelesen werden können.

Information:

- **Wenn auf der USV noch keine Anwendereinstellungen vorhanden sind, wird von der Funktion beim Lesen ein Fehler geliefert.**
- **Lesen Sie die Beschreibung des Module Correction Kommandos (siehe Seite 34) für mehr Informationen zu den USV Anwendereinstellungen.**

```
// Get UPS user settings.
//
// Parameters
//   LowBatShutdownTime
//     [out] Points to a variable that receives the
//     low battery shutdown time in seconds.
//
// Return 0 at success and -1 at failure.
int GetUpsUserSettings(unsigned short *LowBatShutdownTime)
{
    struct
    {
        unsigned char  Ok;
        unsigned char  Special;
        unsigned short Counter;
        unsigned short LowBatShutdownTime;
        unsigned short Reserve;
    } data;
    int retvalue;
    unsigned char ErrorCode;

    retvalue = MtcxReadCommand(MTCX_CMD_MODULE_CORR, 0x600, MTCX_DEVNUM_APC910_IF1, 0,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
         // TODO: add your error handling here
         return -1;
    }

    *LowBatShutdownTime = data.LowBatShutdownTime;
    return 0;
}
```

Listing 92: GetUpsUserSettings – USV Anwendereinstellungen lesen

Das folgende C Codebeispiel zeigt, wie die USV Anwendereinstellungen geschrieben werden können.

Information:

Die USV muss nicht neu gestartet werden, damit geänderte Anwendereinstellungen übernommen werden.

```
// Set UPS user settings.
//
// Parameters
//   LowBatShutdownTime
//     [in] Specifies the low battery shutdown time in seconds: 10 to 1200.
//
// Return 0 at success and -1 at failure.
int SetUpsUserSettings(unsigned short LowBatShutdownTime)
{
    struct
    {
        unsigned short LowBatShutdownTime;
        unsigned short Reserve;
    } data;
    int retvalue;
    unsigned char ErrorCode;

    data.LowBatShutdownTime = LowBatShutdownTime;

    retvalue = MtcxWriteCommand(MTCX_CMD_MODULE_CORR, 0x600, MTCX_DEVNUM_APC910_IF1, 0,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
         // TODO: add your error handling here
         return -1;
    }
    return 0;
}
```

Listing 93: SetUpsBatterySettings – USV Anwendereinstellungen schreiben

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxReadCommand** finden Sie auf Seite 73.
Die aufgerufene Funktion **MtcxWriteCommand** finden Sie auf Seite 74.

7.17.8 USV abschalten (Shutdown)

Die USV kann mit dem **UPS Service** Kommando des MTCX (siehe Seite 39) abgeschaltet werden.

Diese Funktion wird z.B. vom Windows USV Dienst aufgerufen, bevor das System runtergefahren wird. Damit wird vermieden, dass das System weiterhin im USV Betrieb verbleibt und die USV Batterie „ausleert“, wenn Windows nicht ordnungsgemäß runtergefahren wird.

Das folgende C Codebeispiel zeigt, wie die USV abgeschaltet werden kann. Dabei kann eine Verzögerungszeit in Sekunden eingegeben werden.

Vorsicht!

Die angegebene Verzögerungszeit muss länger sein als die Zeit, die vom Betriebssystem zum Runterfahren benötigt wird. Andernfalls kann es zu Datenverlusten kommen. Beachten Sie dabei auch die Low Battery Shutdown Zeit der USV (siehe Seite 155)!

Information:

- **Diese Funktion liefert keinen Fehler, wenn keine USV „verbunden“ ist (siehe Seite 151).**
- **Lesen Sie die Beschreibung des UPS Service Kommando für weitere Informationen zum Abschaltverhalten der USV.**

```
// Shutdown UPS.
//
// Parameters
//   Delay
//     [in] Specifies the minimum amount of time,
//     in seconds, to wait before turning off the
//     UPS power outlets: 10 to 1200.
//
// Return 0 at success and -1 at failure.
int ShutdownUps(unsigned short Delay)
{
    MTCX_UPS_SHUTDOWN_DATA data;
    int retvalue;
    unsigned char ErrorCode;

    data.Reserve = 0;
    data.Delay = Delay;

    retvalue = MtcxWriteCommand(MTCX_CMD_UPS_SERVICE, 0x01, 0, MTCX_DEV_UPS,
        &data, sizeof(data), &ErrorCode);
    if (retvalue != MTCX_OKAY)
    {
         // TODO: add your error handling here
         return -1;
    }
    return 0;
}
```

Listing 94: ShutdownUps – USV abschalten

Die verwendeten Definitionen finden Sie auf Seite 59.
Die aufgerufene Funktion **MtcxWriteCommand** finden Sie auf Seite 74.

8 Abbildungsverzeichnis

9 Tabellenverzeichnis

10 Listingverzeichnis

11 Stichwortverzeichnis

V

W