SIEMENS

MOBY®

C-Bibliothek MOBY API

Programmieranleitung

Erschienen im März 2005

Sicherheitstechnische Hinweise

Dieses Handbuch enthält Hinweise, die Sie zu Ihrer persönlichen Sicherheit sowie zur Vermeidung von Sachschäden beachten müssen. Die Hinweise zu Ihrer persönlichen Sicherheit sind durch ein Warndreieck hervorgehoben, Hinweise zu allgemeinen Sachschäden stehen ohne Warndreieck. Je nach Gefährdungsstufe werden die Warnhinweise in abnehmender Reihenfolge wie folgt dargestellt:

Gefahr

bedeutet, dass Tod, schwere Körperverletzung eintreten **wird**, wenn die entsprechenden Vorsichtsmaßnehmen nicht getroffen werden.

Warnung

bedeutet, dass Tod oder schwere Körperverletzung eintreten **kann**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Vorsicht

mit Warndreieck bedeutet, dass eine leichte Körperverletzung eintreten kann, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Vorsicht

ohne Warndreieck bedeutet, dass ein Sachschaden eintreten kann, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Achtung

bedeutet, dass ein unerwünschtes Ergebnis oder Zustand eintreten kann, wenn der entsprechende Hinweis nicht beachtet wird.

Beim Auftreten mehrerer Gefährdungsstufen wird immer der Warnhinweis zur jeweils höchsten Stufe verwendet. Wenn in einem Warnhinweis mit dem Warndreieck vor Personenschäden gewarnt wird, dann kann im sleben Warnhinweis zusätzlich eine Warnung vor Sachschäden angefügt sein.

Qualifiziertes Personal

Das zugehörige Gerät/System darf nur in Verbindung mit dieser Dokumentation eingerichtet und betreiben werden. Inbetriebsetzung und Betrieb eines Gerätes/Systems dürfen nur von **qualifiziertem Personal** vorgenommen werden. Qualifiziertes Personal im Sinne der sicherheitstechnischen Hinweise dieses Handbuchs sind Personen, die die Berechtigung haben, Geräte, Systeme und Stromkreise gemäß den Standards der Sicherheitstechnik in Betrieb zu nehmen, zu erden und zu kennzeichnen.

Marken

Alle mit dem Schutzrechtsvermerk ® gekennzeichneten Bezeichnungen sind eingetragene Marken der Siemens AG. Die übrigen Bezeichnungen in dieser Schrift können Marken sein, deren Benutzung durch Dritte für deren Zwecke die Rechte der Inhaber verletzten können.

Siemens AG
Automation and Drives
Special Products, Projects Automotive Industry, Training
Postfach 4848, D-90327 Nürnberg

Siemens-Aktiengesellschaft

Bestell-Nr. (4)J31069-D0137-U001-A5-0018

Inhaltsverzeichnis

1 Einleitung

1.1 Produktüberblick

Mit der Produktfamilie MOBY werden verschiedene Identifikationssysteme angeboten, welche für die Steuerung und Optimierung des Materialflusses in der Produktion, Fertigung, Distribution und Logistik weltweit eingesetzt werden.

Die Identifikationssysteme werden aus drei Komponenten gebildet:

- Mobile Datenspeicher (MDS)
- Schreib-/Lesegeräte (SLG), Schreib-/Leseantennen (SLA) oder Serielle Interface Module (SIM)
- Anschaltmodule (ASM)

Diese Komponenten stehen in verschiedenen Ausführungen zur Verfügung.

Für die Kommunikation mit den MOBY-Systemen, die seriell an PC mit Windows 98/NT 4.0/2000/XP oder an Ethernet mit PC mit Windows 98/NT 4.0/2000/XP angekoppelt sind, steht eine einheitliche C-Bibliothek MOBY API für anwenderspezifische MOBY-Applikationen zur Verfügung. Sie bietet eine einfache und schnelle Systemeinbindung.

1.2 Serielle Ankopplung an PC

Die serielle Ankopplung der MOBY-Systeme an PC mit Windows 98/NT 4.0/2000/XP erfolgt über ASM, SIM oder SLG gemäß Tabelle 1-1.

An den MOBY-Komponenten stehen meistens die physikalischen Schnittstellen RS 232 (kurze Kabellängen) und RS 422 (lange Kabellängen) zur Verfügung. Das Protokoll ist immer 3964R (siehe Anhang C).

1.2.1 Seriell anschließbare MOBY-Systeme

Tabelle 1-1 Mit MOBY API betreibbare Komponenten bei serieller Ankopplung an PC

MOBY-Produktfamilie	Anschaltung	zugehörige Dokumentation
MOBY E	ASM 420 mit SLG 7x	/01/
	ASM 424 mit SLG 7x (ein bis vier SLG 7x an einem ASM 424)	/05/
	ASM 724 mit SLA 71 (ein bis vier SLA 71 an einem ASM 724)	/04/
MOBY F	ASM 824 mit SLA 81 (ein bis vier SLA 81 an einem ASM 824)	/03/
MOBY I (ohne Filehandler)	ASM 420 mit SLG 4x	/01/
	ASM 424 mit SLG 4x (ein bis vier SLG 4x an einem ASM 424)	/05/
	SIM 41	/02/

Tabelle 1-1 Mit MOBY API betreibbare Komponenten bei serieller Ankopplung an PC

MOBY-Produktfamilie	Anschaltung	zugehörige Dokumentation
MOBY U (ohne Filehandler)	SLG U92	/06/

1.2.2 C-Bibliothek MOBY API für serielle Ankopplung an PC

Das Produkt C-Bibliothek MOBY API für anwenderspezifische MOBY-Applikationen ist ablauffähig unter Windows 98/NT 4.0/2000/XP und besteht für die Realisierung und den Ablauf von MOBY-Applikationen aus den Komponenten:

- C-Bibliothek MOBY_API.LIB
- MOBY-Headerdatei MOBY_API.H
- dynamische Link-Bibliothek MOBY_API.DLL
- Treiber 3964R 3964R.DLL
- Konfigurationsprogramm für 3964R CPL3964R.CPL
- 3964R-Headerdatei 3964R.H
- Beispielapplikation
  - ablauffähig EXAMPLE.EXE
  - Sourcecode EXAMPLE.CPP

Die Steuerung der Schreib-/Lesegeräte (SLG), der Schreib-/Leseantennen (SLA) und der Seriellen Interface Module (SIM) erfolgt über die serielle Schnittstelle durch Übertragung strukturierter Telegramme mit dem Protokoll 3964R. Das Protokoll 3964R liegt als DLL (Dynamic Link Library) vor.
Die Telegramme werden durch Bibliotheksfunktionen generiert bzw. empfangen. Die Bibliotheksfunktionen setzen wiederum auf den vorhandenen Treiber 3964R auf und stellen die Schnittstelle zwischen den MOBY-Applikationen und den MOBY-Systemen dar.

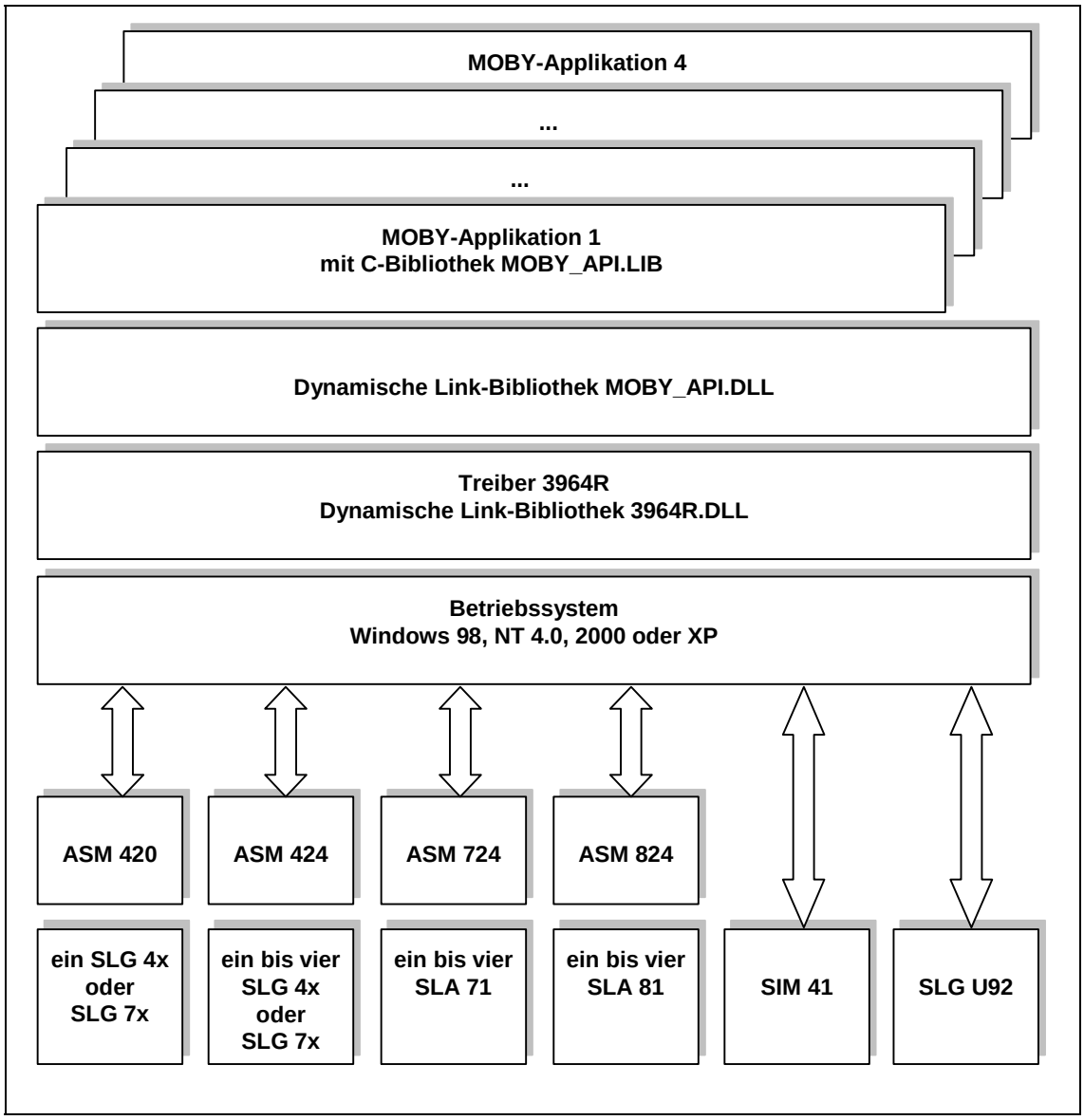


Bild 1-1 Struktur MOBY API für serielle Ankopplung an PC

Die Funktionen stehen als DLL (Dynamic Link Library) in der dynamischen Link-Bibliothek MOBY_API.DLL und die Funktionsaufrufe in der C-Bibliothek MOBY_API.LIB zur Verfügung:

- je ASM, SIM oder SLG U92 maximal eine Applikation
- eine Link-Bibliothek für maximal vier Applikationen

Die Treiberfunktionalität ist in Form einer „CPL“-Datei unter der Systemsteuerung von Windows zu konfigurieren.

- Konfigurationsprogramm CPL3964R.CPL

In jede MOBY-Applikationen ist eine Headerdatei mit Funktionsdeklarationen einzubinden.

- Headerdatei MOBY_API.H

Die Headerdatei MOBY_API.H muss in den Sourcecode der Applikation mit dem Präprozessorbefehl „#include“ eingebunden werden. Dadurch werden alle Funktionsaufrufe und Konstanten deklariert. Die Headerdatei wurde mit und für Visual C++ entwickelt. Falls eine andere Programmiersprache bzw. ein C++-Dialekt einer anderen Firma verwendet werden soll, kann es sein, dass die Headerdatei angepasst werden muss.
Die Headerdatei MOBY_API.H benötigt wiederum die Include-Datei 3964R.H von dem Treiber 3964R.

Beispielapplikation

Über die C-Bibliothek hinaus wird für den leichten Einstieg in die Realisierung einer Anwenderapplikation eine ablauffähige Beispielapplikation EXAMPLE.EXE mitgeliefert. Sie steht zusätzlich im Sourcecode (EXAMPLE.CPP) zur Verfügung. Sie kann als Basis für eine zu erstellende Anwenderapplikation verwendet werden und ist voll link- und ablauffähig.

1.2.3 Systemvoraussetzungen für serielle Ankopplung an PC

Für die Anwendung der C-Bibliothek unter Windows™ müssen folgende Voraussetzungen erfüllt sein:

• Personalcomputer (PC)	AT-kompatibler PC oder PG
• Betriebssystem	Windows™ 98/NT 4.0/2000/XP
• Freie serielle Schnittstelle(n)	RS 232/RS 422
• Programmiervorschriften	Die Interfacebibliothek MOBY_API.LIB ist in „C“ erstellt und kompatibel zum Microsoft Visual C++ Compiler Version ≥ 6.0. Andere Programmiersprachen (z. B. Visual Basic) mittels eines Wrapper.

1.3 Ankopplung an Ethernet

Die Ankopplung der MOBY-Systeme an Ethernet mit PC mit Windows 98/NT 4.0/ 2000/XP erfolgt über das Anschaltmodul ASM 480 gemäß Tabelle 1-2. Das MOBY-System wird seriell an das ASM 480 angeschlossen. Das Protokoll zwischen dem ASM 480 und dem MOBY-System ist immer 3964 R. Das ASM 480 dient als Protokollumsetzer zwischen dem TCP/IP-Protokoll und dem Protokoll 3964R.

1.3.1 Anschließbare MOBY-Systeme an Ethernet

Tabelle 1-2 Mit MOBY API betreibbare Komponenten bei Ankopplung an Ethernet

MOBY-Produktfamilie	Anschaltung	zugehörige Dokumentation
MOBY U (ohne Filehandler)	ASM 480 mit SLG U92	/06/

1.3.2 C-Bibliothek MOBY API für Ankopplung an Ethernet

Das Produkt C-Bibliothek MOBY API für anwenderspezifische MOBY-Applikationen ist ablauffähig unter Windows 98/NT 4.0/2000/XP und besteht für die Realisierung und den Ablauf von MOBY-Applikationen aus den Komponenten:

- C-Bibliothek MOBY_API_T.LIB
- MOBY-Headerdatei MOBY_API_T.H
- dynamische Link-Bibliothek MOBY_API_T.DLL
- Beispielapplikationen
  - ablauffähig MOBY API.EXE
    (in EXAMPLE.ZIP, Unterverzeichnis „MOBY_API_T“ abgelegt)
  - Sourcecode der ablauffähigen Beispielapplikation MOBY API.EXE
    (in EXAMPLE.ZIP, Unterverzeichnis „MOBY_API_T“ abgelegt)
  - Sourcecode für eine einfache Beispielapplikation
    (in EXAMPLE.ZIP, Unterverzeichnis „TEST“ abgelegt)

Die Steuerung der Schreib-/Lesegeräte (SLG) erfolgt an Ethernet durch Übertragung strukturierter Telegramme, die an der seriellen Schnittstelle des SLG mit dem Protokoll 3964R übertragen werden.
Die Telegramme werden durch Bibliotheksfunktionen generiert bzw. empfangen. Die Bibliotheksfunktionen setzen wiederum auf den Treiber TCP/IP (dynamische Link-Bibliothek) auf und stellen die Schnittstelle zwischen den MOBY-Applikationen und den MOBY-Systemen dar.

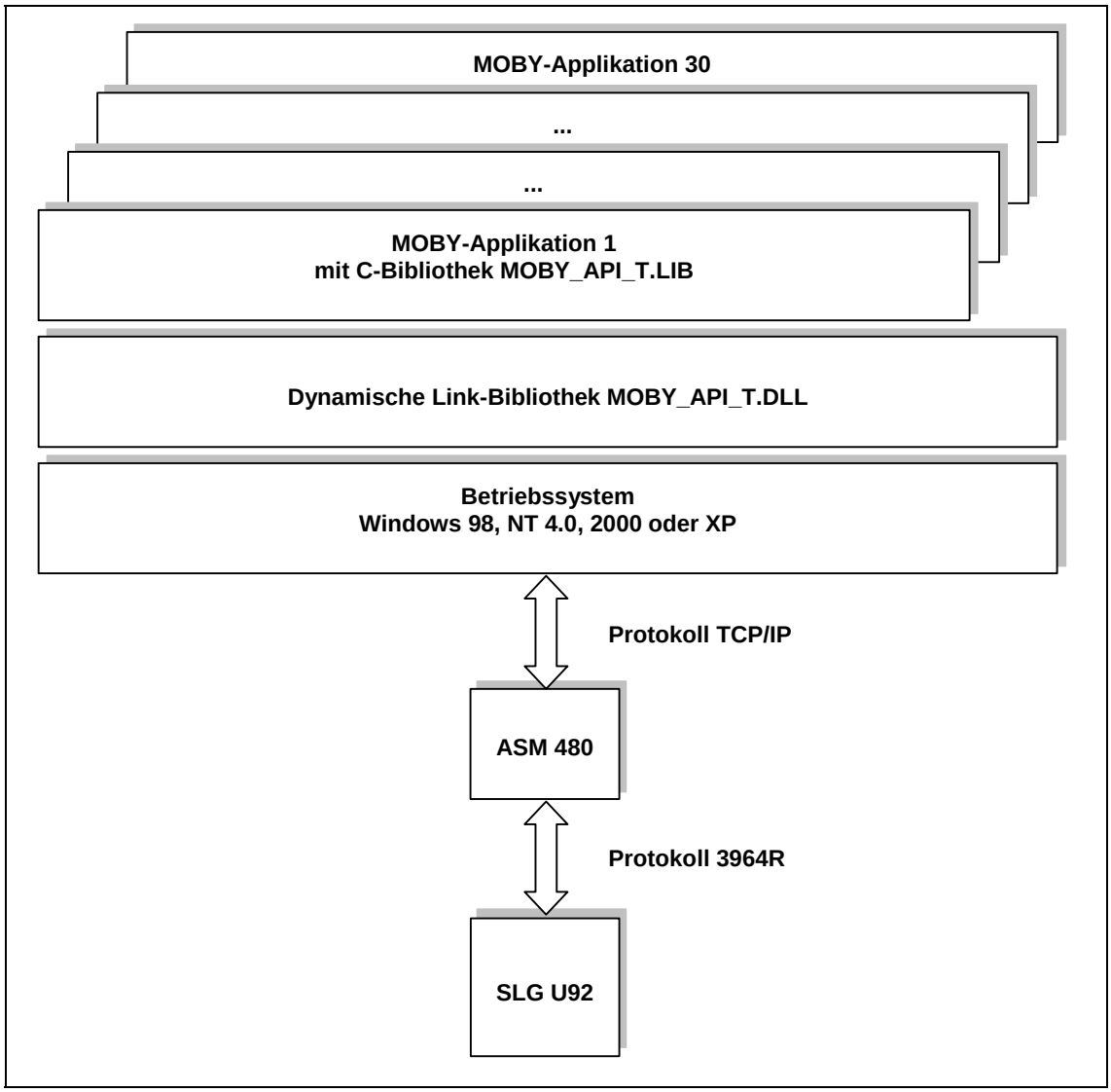


Bild 1-2 Struktur MOBY API bei Ankopplung an Ethernet

Die Funktionen stehen als DLL (Dynamic Link Library) in der dynamischen Link-Bibliothek MOBY_API_T.DLL und die Funktionsaufrufe in der C-Bibliothek MOBY_API_T.LIB zur Verfügung:

- je SLG U92 maximal eine Applikation
- eine Link-Bibliothek für maximal 30 Applikationen

In jede MOBY-Applikation ist eine Headerdatei mit Funktionsdeklarationen einzubinden.

- Headerdatei MOBY_API_T.H

Die Headerdatei MOBY_API_T.H muss in den Sourcecode der Applikation mit dem Präprozessorbefehl „#include“ eingebunden werden. Dadurch werden alle Funktionsaufrufe und Konstanten deklariert. Die Headerdatei wurde mit und für Visual C++ entwickelt. Falls eine andere Programmiersprache bzw. ein C++-Dialekt einer anderen Firma verwendet werden soll, kann es sein, dass die Headerdatei angepasst werden muss.

Beispielapplikationen

Über die C-Bibliothek hinaus werden für den leichten Einstieg in die Realisierung einer Anwenderapplikation zwei Beispielapplikationen mitgeliefert:

- ablauffähige Applikation MOBY API.EXE zum Bedienen von bis zu vier SLG U92 mit ASM 480, die auch im Sourcecode zur Verfügung steht
- einfache Beispielapplikation TEST.CPP im Sourcecode (VC++ 6.0) als Basis für eine zu erstellende Anwenderapplikation für ein SLG U92 an ASM 480. Sie dient zur übersichtlichen Darstellung der wichtigsten MOBY API-Funktionen.

  Siehe hierzu auch Kapitel 5.2.

Beide Beispielapplikationen sind in der Datei EXAMPLE.ZIP enthalten.

1.3.3 Systemvoraussetzungen für Ankopplung an Ethernet

Für die Anwendung der C-Bibliothek unter Windows™ müssen folgende Voraussetzungen erfüllt sein:

• Personalcomputer (PC)	AT-kompatibler PC oder PG
• Betriebssystem	Windows™ 98/NT 4.0/2000/XP
• LAN-Anschluss am PC	
• Programmiervorschriften	Die Interfacebibliothek MOBY_API_T.LIB ist in „C“ erstellt und kompatibel zum Microsoft Visual C++ Compiler Version ≥ 6.0. Andere Programmiersprachen (z. B. Visual Basic) mittels eines Wrapper.

2 Installation

2.1 Auslieferungsdateien

- mobyapi.msi Windows Installer Package mit C-Bibliothek für MOBY-Applikationen für serielle Ankopplung an PC
  Mit dem Windows-Installationspaket (mobyapi.msi) installieren oder deinstallieren Sie die C-Bibliothek für MOBY-Applikationen.
- mobyapi_t.msi Windows Installer Package mit C-Bibliothek für MOBY-Applikationen für Ankopplung an Ethernet
  Mit dem Windows-Installationspaket (mobyapi_t.msi) installieren oder deinstallieren Sie die C-Bibliothek für MOBY-Applikationen.
- InstMsiW.exe Windows Installer für Windows NT
- InstMsiA.exe Windows Installer für Windows 98

Achtung

Wenn das Windows-Installationspaket mobyapi.msi oder mobyapi_t.msi bei Windows NT oder Windows 98 nicht als Typ „Windows Installer Package“ angezeigt wird, muss der Windows Installer aktiviert werden.

Die Installation der C-Bibliothek MOBY API läuft je nach Ankopplung (Schnittstellenvariante) in einem eigenen Installationszweig ab:

- Installation für serielle Ankopplung an PC: mobyapi.msi
- Installation für Ankopplung an Ethernet: mobyapi_t.msi

2.2 Installation der C-Bibliothek MOBY API für serielle Ankopplung an PC

Nach dem Start des Windows-Installationspakets mobyapi.msi werden Sie durch die Installation geführt.

1. Das Paket mobyapi.msi starten

   Bei der erstmaligen Installation der C-Bibliothek wird das Fenster „Installation der C-Bibliothek aktivieren“ (Bild 2-1) geöffnet, sonst erscheint das Fenster „Bestehende C-Bibliothek MOBY API überschreiben oder deinstallieren“ (Bild 2-6).

Achtung

Die C-Bibliothek MOBY API sollte nie manuell teilweise oder komplett installiert oder deinstalliert werden. Dadurch kann es zu Problemen bei der Installation oder Deinstallation mit mobyapi.msi kommen.

Neuinstallation

2. Installation der C-Bibliothek aktivieren

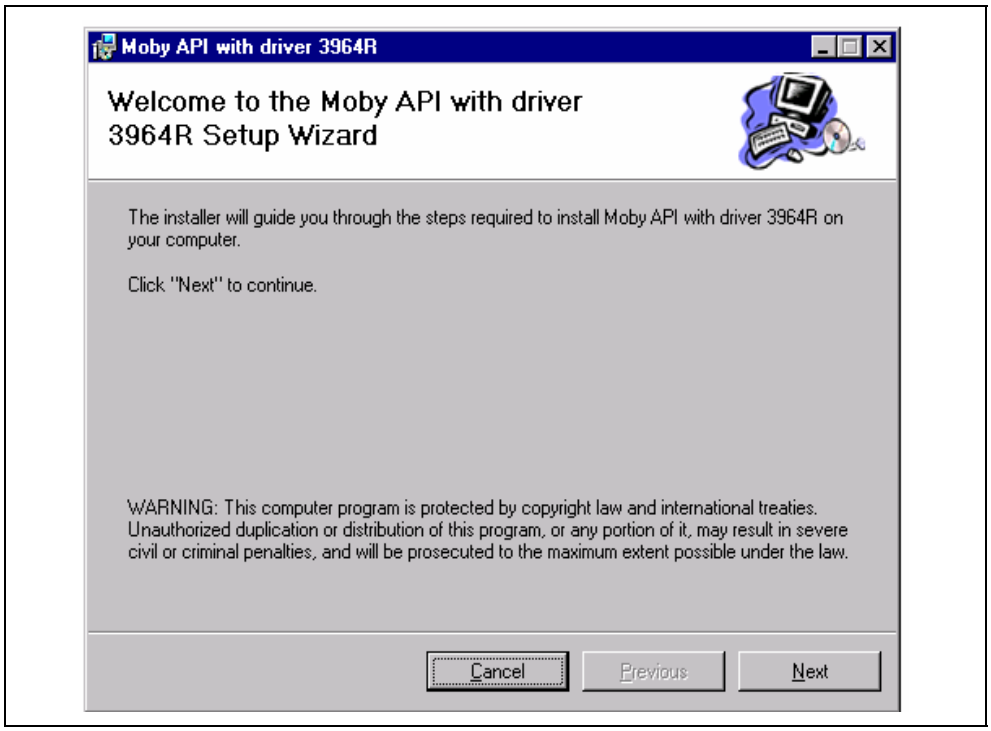


Bild 2-1 Installation der C-Bibliothek aktivieren (serielle Ankopplung an PC)

Mit „Next“ wird die Installation aktiviert und das Fenster für „Verzeichnis für die C-Bibliothek MOBY API auswählen“ (Bild 2-2) ausgegeben.
Mit „Cancel“ wird die Installation wieder abgebrochen (mit vorheriger Bestätigung).

3. Verzeichnis für die C-Bibliothek MOBY API auswählen

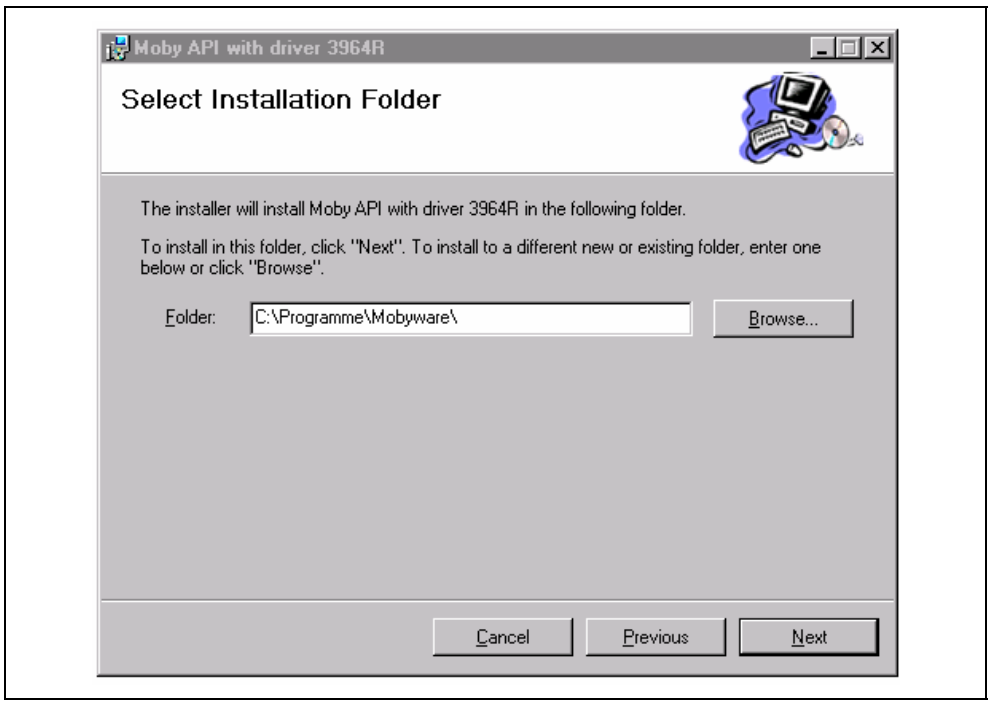


Bild 2-2 Verzeichnis für die Installation auswählen (serielle Ankopplung an PC)

Es erscheint das Fenster „Select Installation Folder“ mit der Standardeinstellung „C:\Programme\mobyware“. Mit „Next“ kann diese Einstellung übernommen, im Feld Folder kann sie abgeändert oder mit Browse ein vorhandenes Verzeichnis (siehe Bild 2-3) ausgewählt werden.

Mit „Next“ wird die Einstellung übernommen, und es erscheint das Fenster „Confirm Installation“ (Bild 2-4).

4. Ein vorhandenes Verzeichnis auswählen

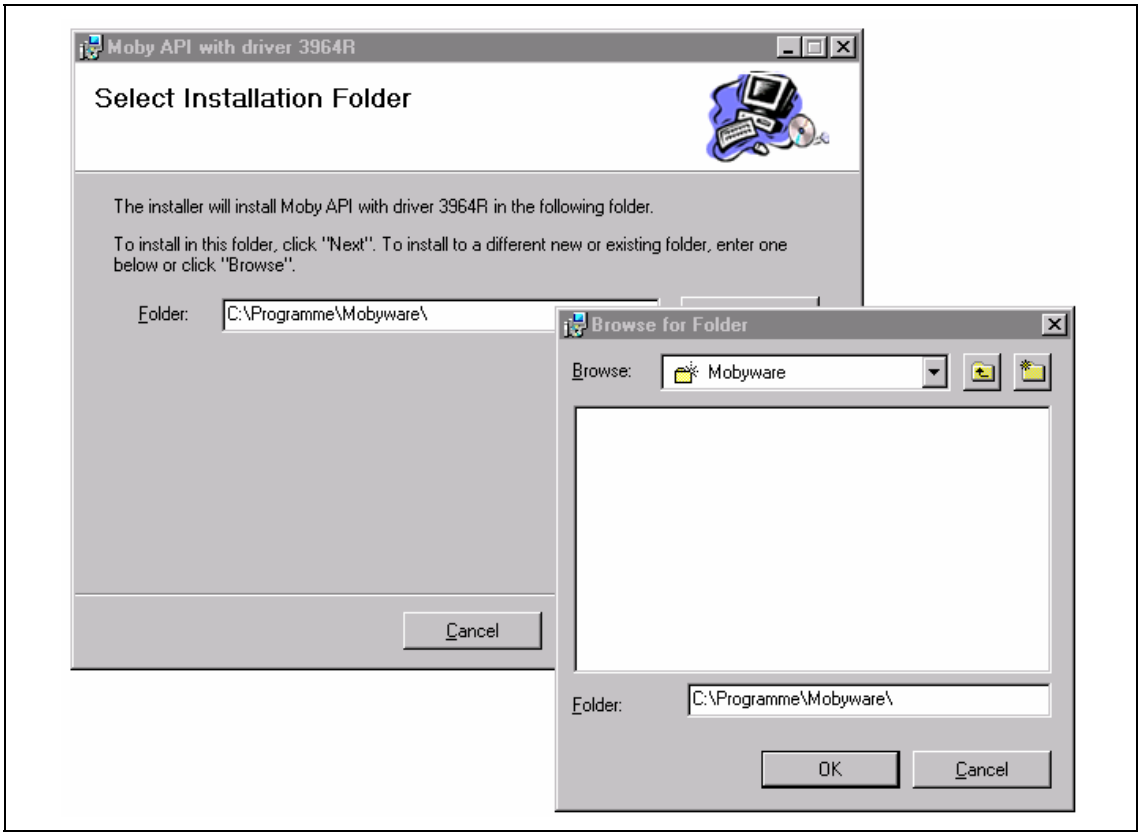


Bild 2-3 Vorhandenes Verzeichnis auswählen (serielle Ankopplung an PC)

Im Fenster „Browse for Folder“ kann ein bestehendes Verzeichnis ausgewählt, in das Feld Folder übernommen und geändert werden.

- Mit OK erfolgt die Rückkehr in das Fenster „Select Installation Folder“ mit der Übernahme des Verzeichnisses.
- Mit Cancel erfolgt die Rückkehr. Dabei wird die Auswahl verworfen.

5. Die Installation der Software MOBY API ausführen

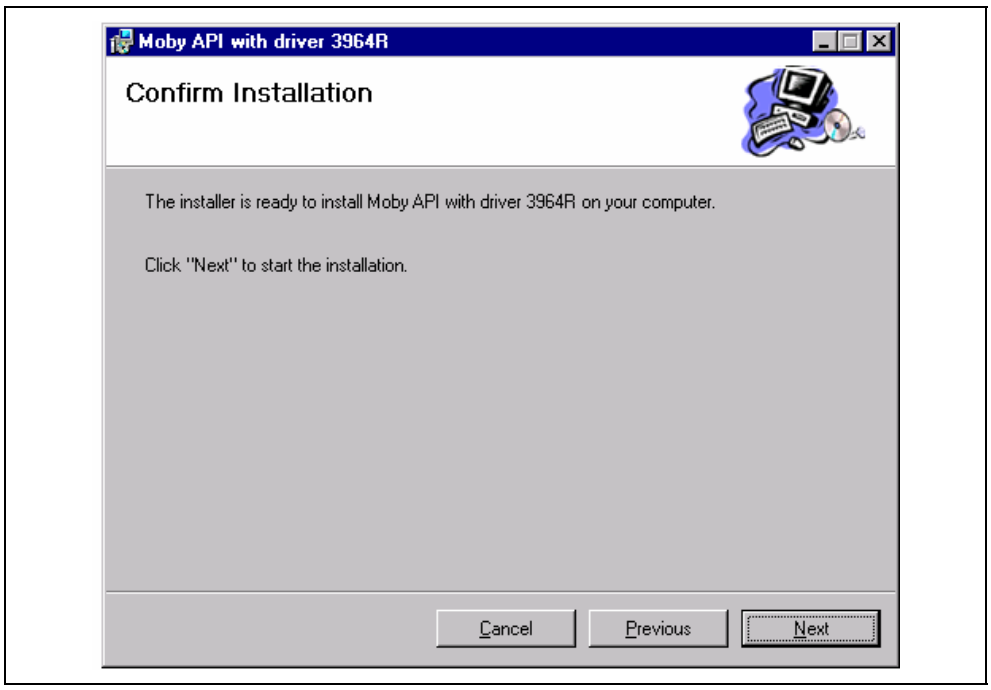


Bild 2-4 Installation ausführen (serielle Ankopplung an PC)

Mit „Next“ wird die Software MOBY API für die Erstellung von Applikationen unter den nachfolgend aufgeführten Unterverzeichnissen innerhalb des ausgewählten Verzeichnisses abgelegt:

\example	example.cpp	Beispielapplikation im Sourcecode
	example.exe	Beispielapplikation als ablauffähiges Programm
\include	3964R.H	Include-Datei vom Treiber 3964R für die Headerdatei MOBY_API.H
	MOBY_API.H	Headerdatei für Anwenderapplikationen
\lib	MOBY_API.LIB	C-Bibliothek für Anwenderapplikationen

Die DLL von dem Treiber 3964R und der dynamischen Link-Bibliothek MOBY_API und das Konfigurationsprogramm CPL3964R.CPL werden in Abhängigkeit des Windows-Systems unter folgendes Verzeichnis kopiert:

- Windows NT 4.0: C:\WINNT\SYSTEM32
- Windows 98: C:\WIN98\SYSTEM
- Windows 2000: C:\WIN2000\SYSTEM32
- Windows XP C:\WINNT\SYSTEM32

Für die Nutzung der Beispielapplikation oder den Test einer Anwenderapplikation muss der Treiber 3964R konfiguriert werden (siehe Kapitel 2.2.2). Die Treiber-DLL erhält ihre Information über vorhandene Schnittstellen und deren Konfiguration mit Hilfe der Windows-Registry. Die Einträge in die Registry für die Benutzung des Treibers 3964R werden während der Installation automatisch generiert.

6. Vorgang „Bestehende C-Bibliothek MOBY API installieren“ ist beendet

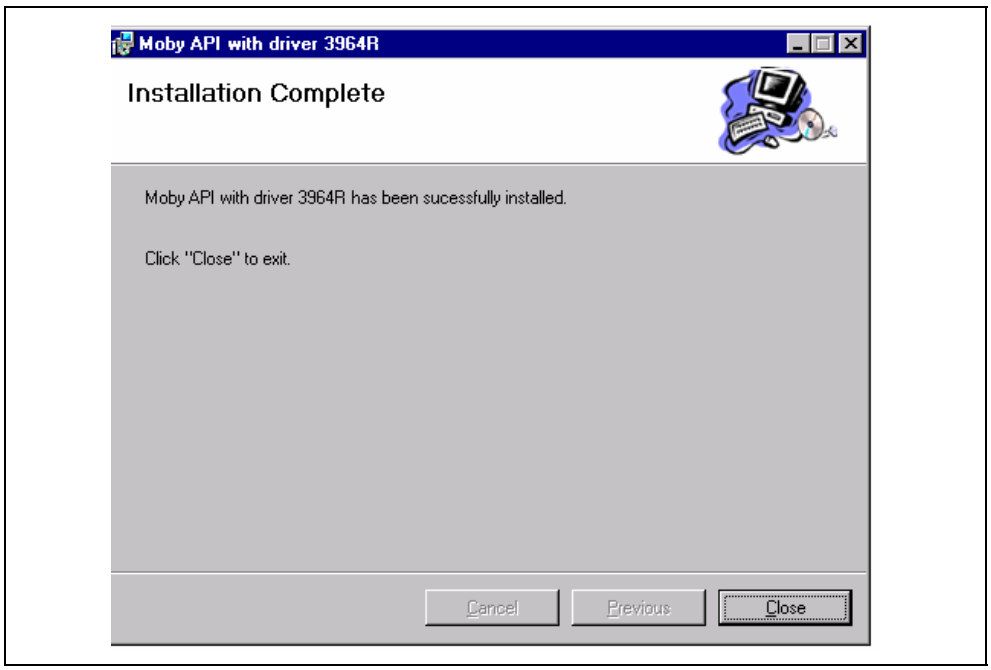


Bild 2-5 Installation vollständig (serielle Ankopplung an PC)

Mit Close wird der Dialog beendet.

Bestehende C-Bibliothek überschreiben

7. Bestehende C-Bibliothek MOBY API überschreiben oder deinstallieren

Bild 2-6 Bestehende C-Bibliothek überschreiben (serielle Ankopplung an PC)

Wenn die bestehende C-Bibliothek MOBY API überschrieben werden soll, ist „Repair MOBY API with driver 3964R“ zu selektieren und anschließend mit Finish der Vorgang zu starten (siehe Bild 2-7).

Achtung

Achten Sie darauf, dass Sie die neuen Headerdateien 3964R.H und MOBY_API.H und die C-Library MOBY_API.LIB in das Entwicklungsprojekt übernehmen.

8. Bestehende C-Bibliohtek MOBY API überschreiben

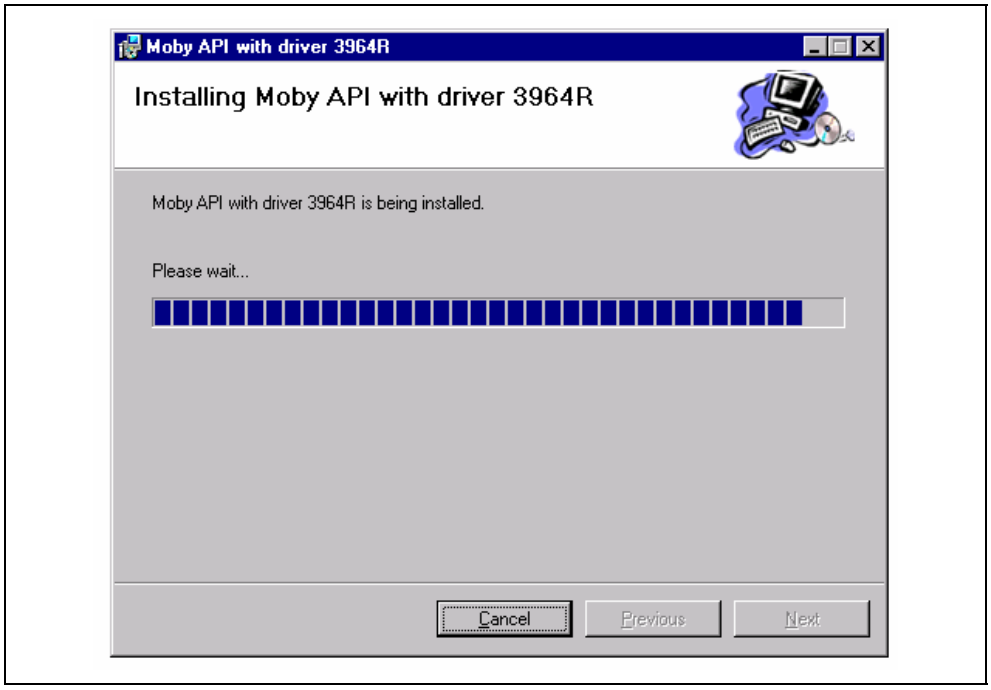


Bild 2-7 Fenster während des Installationsvorgangs „Überschreiben“ (serielle Ankopplung an PC)

Die bestehende C-Bibliothek MOBY API wird überschrieben.

9. Vorgang „Bestehende C-Bibliothek MOBY API überschreiben“ ist beendet

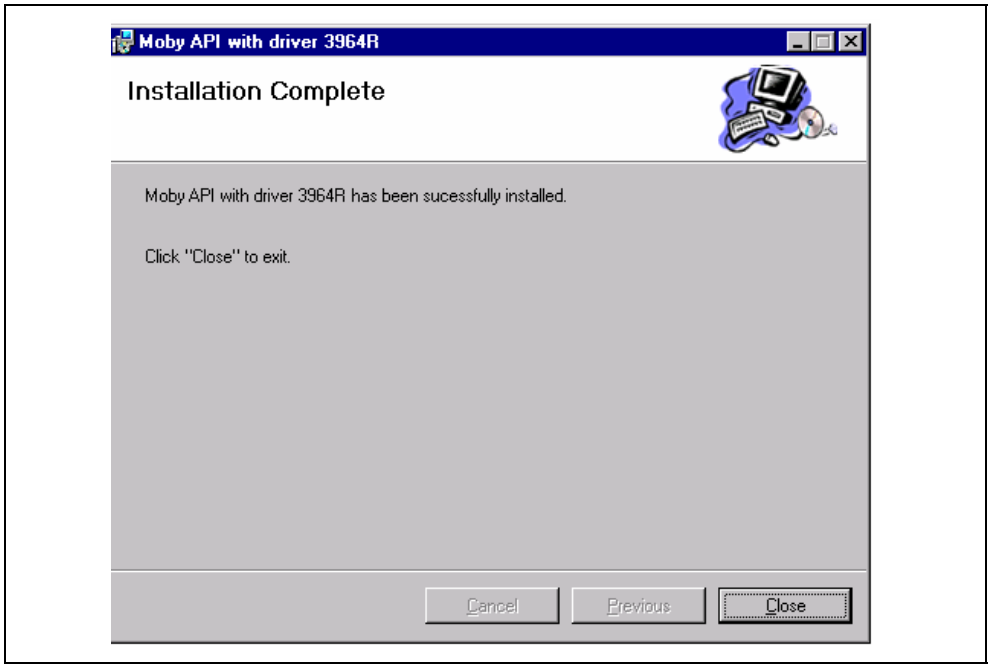


Bild 2-8 „Überschreiben“ ist vollständig (serielle Ankopplung an PC)

Mit Close wird der Dialog beendet.

Deinstallation

Wenn die bestehende C-Bibliothek MOBY API deinstalliert werden soll, ist „Remove MOBY API with driver 3964R“ zu selektieren (siehe Bild 2-6) und anschließend mit Finish der Vorgang zu starten (siehe Bild 2-9).

10. Bestehende C-Bibliothek MOBY API deinstallieren

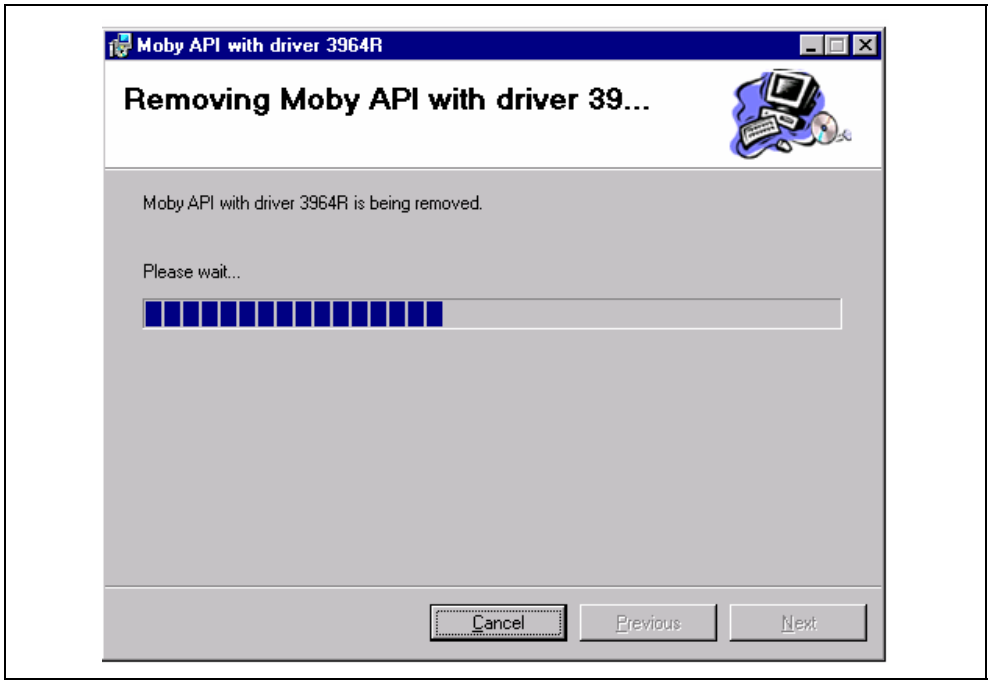


Bild 2-9 Fenster während der Deinstallation (serielle Ankopplung an PC)

11.Vorgang „Bestehende C-Bibliothek MOBY API deinstallieren“ ist beendet

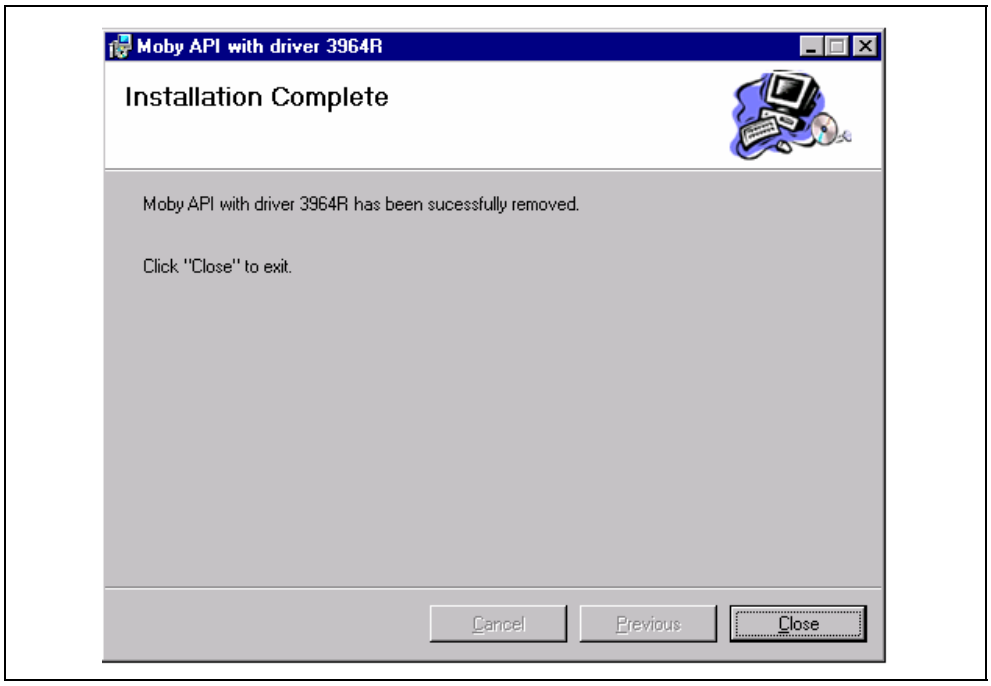


Bild 2-10 Deinstallation ist vollständig (serielle Ankopplung an PC)

Mit Close wird der Dialog beendet.

2.2.1 Anpassung der Registry

Bei Windows 2000, Windows NT und Windows XP besteht eine Besonderheit: Wenn der Treiber vom Administrator/Ersteller-Besitzer installiert wurde und ein User die Berechtigung für den 3964R-Treiber erhalten möchte, sind die nachfolgend aufgeführten Bedienungsschritte vom Administrator/Ersteller-Besitzer auszuführen.

1. Das Programm REGEDT32.EXE starten.
2. Im Fenster HKEY_LOCAL_MACHINE (siehe Bild 2-11) den Pfad „Siemens-741“ aufschlagen.

Bild 2-11 Fenster HKEY_LOCAL_MACHINE

3. In der Menüleiste „Sicherheit“ anwählen (siehe Bild 2-11). Es erscheint das Fenster Registrierungsschlüsselberechtigungen. In diesem Fenster die Option „Berechtigungen...“ einschalten und den User auswählen, für den die Berechtigung hinzugefügt werden soll (siehe Bild 2-12).

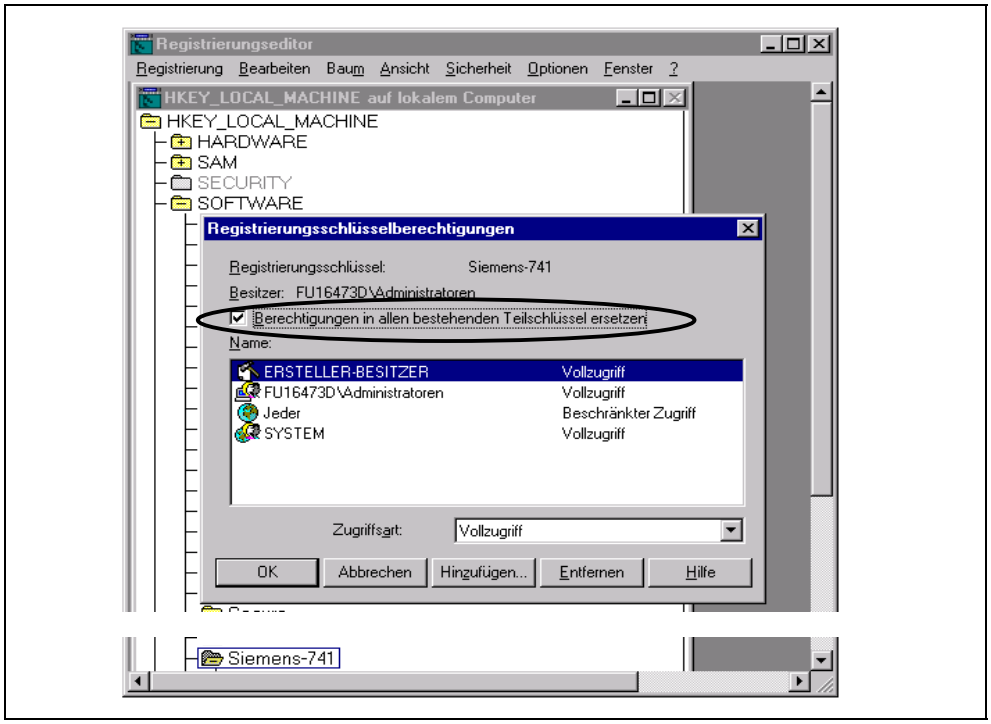


Bild 2-12 Fenster Registrierungsschlüsselberechtigungen

4. Die Schaltfläche „Hinzufügen...“ anwählen. Im jetzt erscheinenden Fenster bei „Zugriffsart“ auf „Vollzugriff“ umstellen (siehe Bild 2-13). Anschließend mit OK alle Fenster schließen.

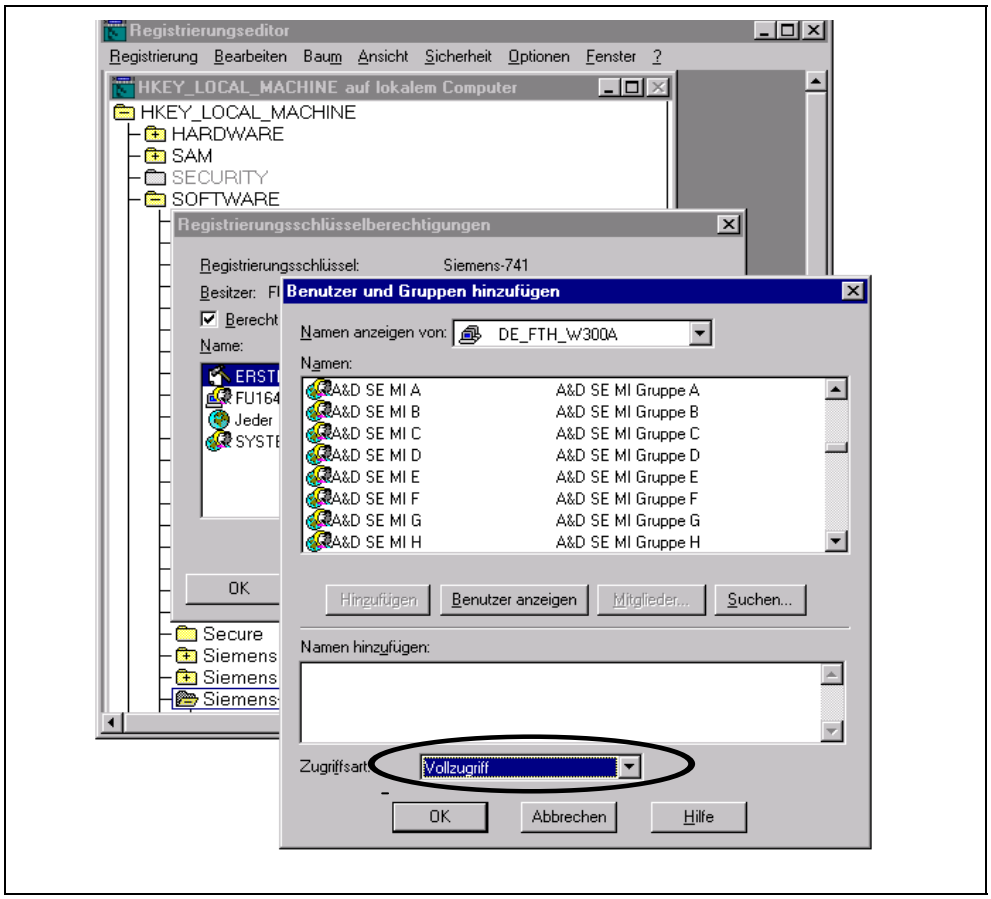


Bild 2-13 Fenster Benutzer und Gruppen hinzufügen

2.2.2 Konfiguration des Treibers 3964R

Die Konfiguration der Funktionalität des Treibers 3964R ist mit dem Konfigurationsprogramm CPL3964R.CPL durchzuführen (siehe Bild 2-14) und unter der Systemsteuerung von Windows aufzurufen.

Folgende Parameter sind als Standardwerte über das Konfigurationsprogramm einzustellen:

- Data bits 8
- Stop bits 1
- Parity Odd
- Send buffer 255
- Receive buffer 255
- Discard conflict telegrams √ (wenn der Treiber 3964R Slave ist)

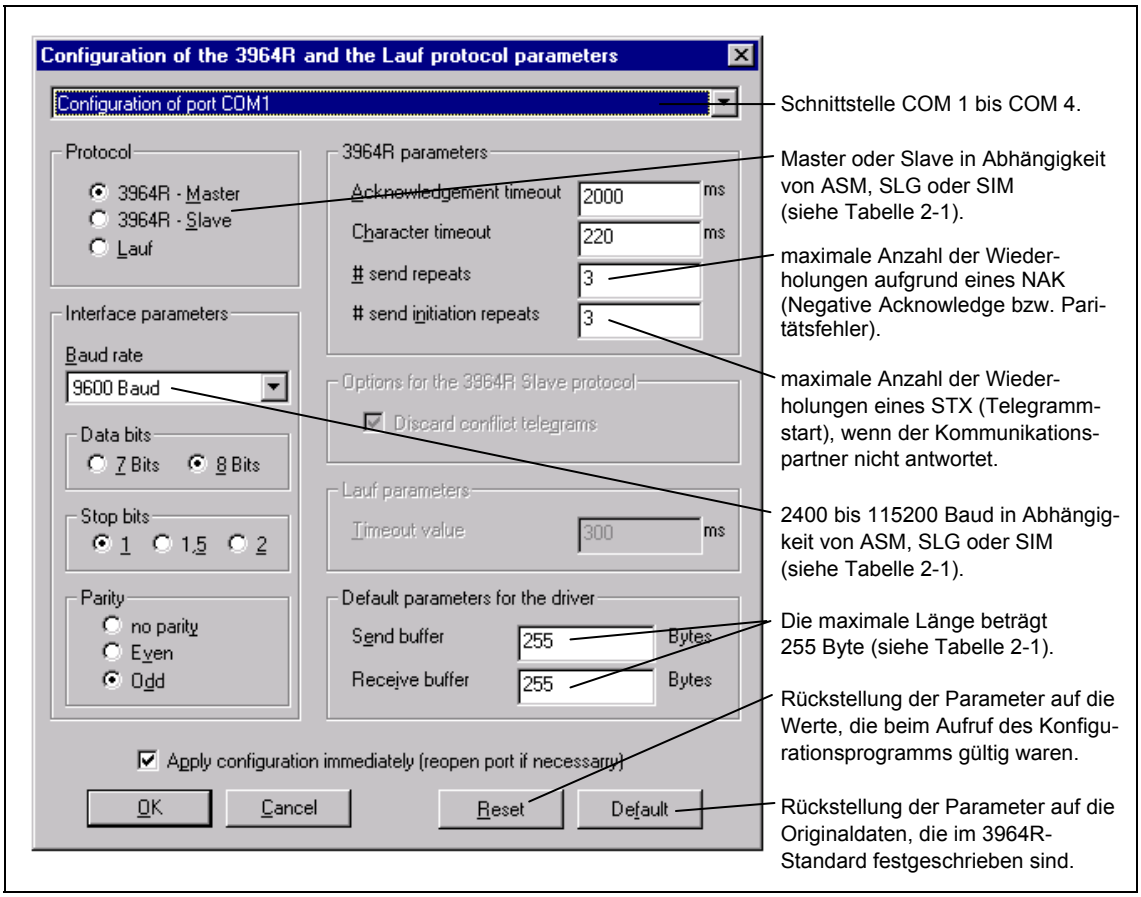


Bild 2-14 Dialog zur Konfiguration des Protokolls 3964R

Achtung

Bei MOBY U sind standardmäßig die Zeiten Acknowledgement timeout mit 150 ms und Character timeout mit 50 ms parametriert.

Achtung

Ist die Checkbox „Apply configuration immediately“ aktiviert, so werden alle geänderten Daten sofort in die Konfiguration übernommen, auch wenn dazu ein Schließen und Öffnen des Ports nötig ist (z. B. bei Änderung der Baudrate). Andernfalls werden nur die Daten übernommen, die keine Schnittstellenparameter berühren (z. B. Timeout-Werte).

Tabelle 2-1 SLG-, ASM- oder SIM-abhängige Treiberparameter

ASM- oder SIM-abhängiger Treiberparameter	SLG/ASM/SIM					
	ASM 824	**ASM 724**	**ASM 424**	**ASM 420**	**SIM 41**	**SLG U92**
Protocol	3964R-Slave	3964R-Slave	3964R-Slave	3964R-Master oder -Slave	3964R-Master oder -Slave	3964R-Slave
Baud rate	9600, 19200 oder 38400 Baud			2400, 4800, 9600, 19200 oder 38400 Baud	2400, 4800 oder 9600 Baud	19200, 38400, 57600 oder 115200 Baud
Send buffer	200 Byte	242 Byte	242 Byte	255 Byte	255 Byte	255 Byte
Receive buffer						

Die Größe für Send buffer und Receive buffer wird durch den Telegrammkopf und die befehlsspezifischen Daten bestimmt.

Die Telegrammkopflänge beträgt:

- 4 Byte bei ASM 824, ASM 724 und ASM 424 (siehe Anhang A.5)
- 3 Byte bei SLG U92, ASM 420 und SIM 41.

Achtung

Bei der Datenübertragung darf die MOBY-Nutzdatenlänge die maximale Größe von 248 Byte nicht überschreiten. D.h., die maximale Länge der zu lesenden Daten vom MDS oder der zu schreibenden Daten auf den MDS beträgt 248 Byte (siehe Kapitel 3.4).

2.2.3 Prioritätsvergabe für den Treiber 3964R

Das Betriebssystem Windows darf bei der Kommunikation zwischen dem PC und dem SLG, ASM oder SIM die Zeiten Acknowledgement timeout und Character timeout nicht überschreiten, da sonst die Kommunikation abgebrochen wird und neu aufzusetzen ist. In Abhängigkeit der CPU-Belastung und Datenspeicherzugriffe kann die Kommunikation beeinträchtigt werden. Das heißt, die oben genannten Zeiten werden überschritten. Um dieses Problem zu umgehen oder zu minimieren, wird bei der Installation des C-Interface MOBY API die Thread-Priorität für die Kommunikation auf den Wert 2 eingestellt.

Thread-Priorität 2 bedeutet:	Maximale Priorität nur während der Kommunikation. Diese Einstellung wird für jede vom Treiber 3964R belegte COM-Schnittstelle vor dem Öffnen der COM-Schnittstelle durchgeführt.

Wenn Sie die Thread-Priorität wieder zurücksetzen oder ändern möchten, so müssen Sie die Einstellung über die Registry mit dem folgenden Schlüssel durchführen:

\\ HKEY_LOCAL_MACHINE\Software\Siemens-741\3964r\COMx\ThreadPriority

2.3 Installation der C-Bibliothek MOBY API für Ankopplung an Ethernet

Nach dem Start des Windows-Installationspakets mobyapi_t.msi werden Sie durch die Installation geführt.

1. Das Paket mobyapi_t.msi starten

   Bei der erstmaligen Installation der C-Bibliothek wird das Fenster „Installation der C-Bibliothek aktivieren“ (Bild 2-15) geöffnet, sonst erscheint das Fenster „Bestehende C-Bibliothek MOBY API überschreiben oder deinstallieren“ (Bild 2-20).

Achtung

Die C-Bibliothek MOBY API sollte nie manuell teilweise oder komplett installiert oder deinstalliert werden. Dadurch kann es zu Problemen bei der Installation oder Deinstallation mit mobyapi_t.msi kommen.

Neuinstallation

2. Installation der C-Bibliothek aktivieren

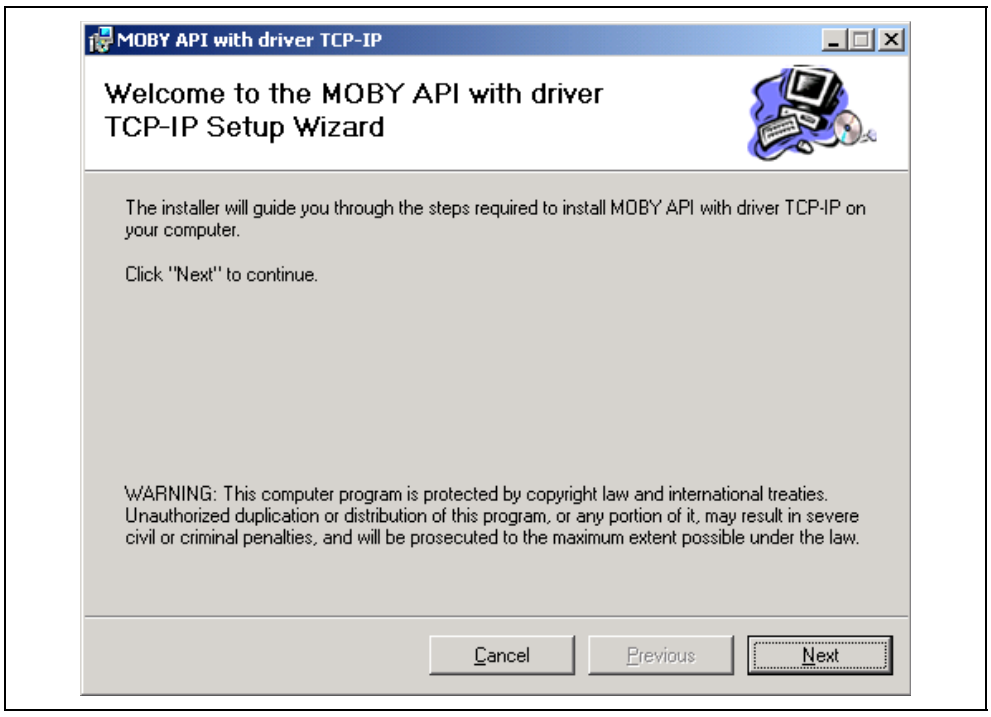


Bild 2-15 Installation der C-Bibliothek aktivieren (Ankopplung an Ethernet)

Mit „Next“ wird die Installation aktiviert und das Fenster für „Verzeichnis für die C-Bibliothek MOBY API auswählen“ (Bild 2-16) ausgegeben.
Mit „Cancel“ wird die Installation wieder abgebrochen (mit vorheriger Bestätigung).

3. Verzeichnis für die C-Bibliothek MOBY API auswählen

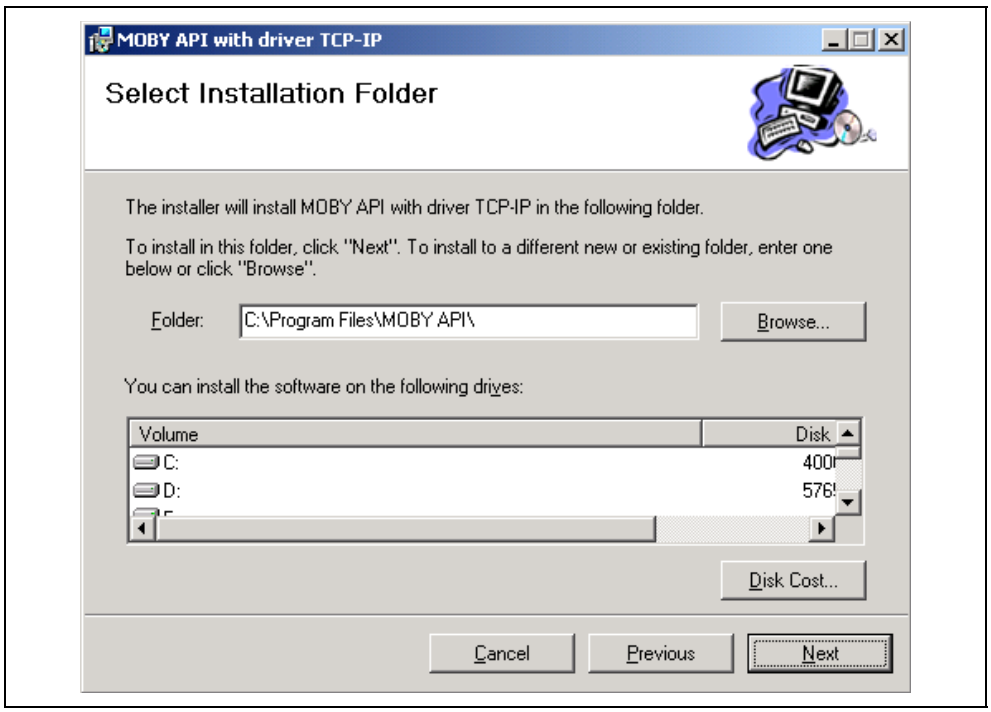


Bild 2-16 Verzeichnis für die Installation auswählen (Ankopplung an Ethernet)

Es erscheint das Fenster „Select Installation Folder“ mit der Standardeinstellung „C:\Program Files\MOBY API“. Mit „Next“ kann diese Einstellung übernommen, im Feld Folder kann sie abgeändert oder mit Browse ein vorhandenes Verzeichnis (siehe Bild 2-17) ausgewählt werden.

Mit „Next“ wird die Einstellung übernommen, und es erscheint das Fenster „Confirm Installation“ (Bild 2-18).

4. Ein vorhandenes Verzeichnis auswählen

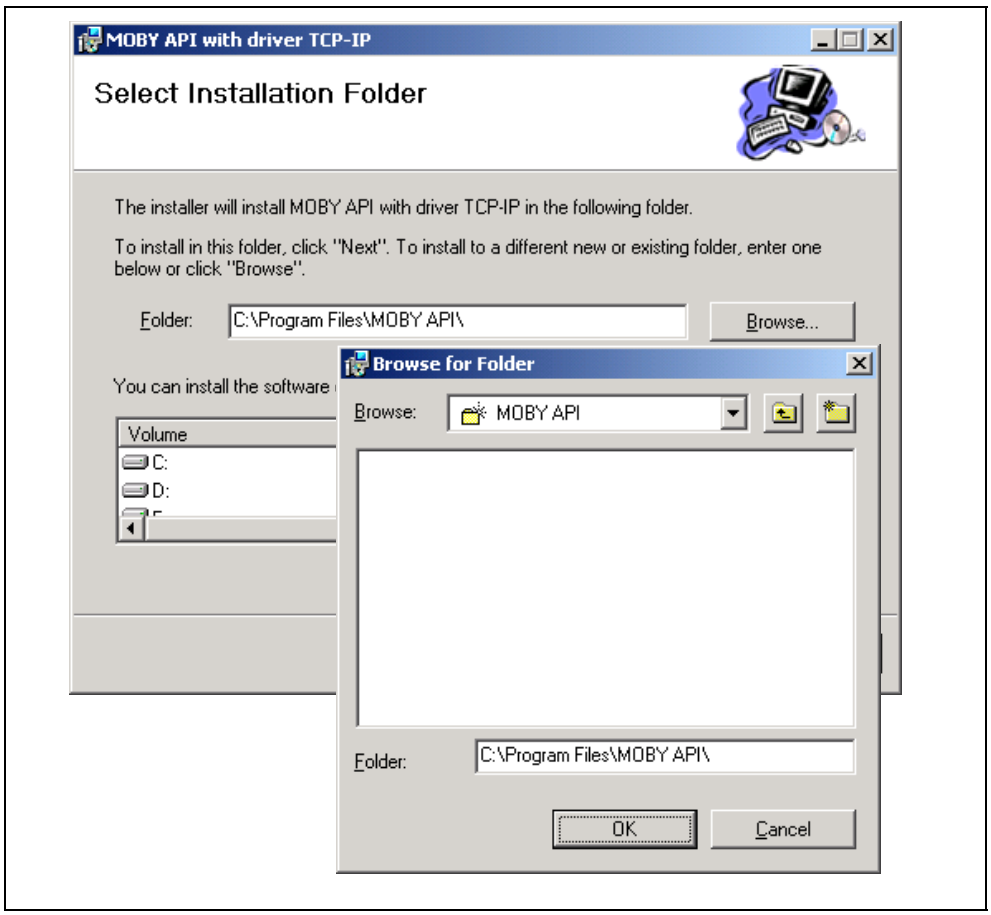


Bild 2-17 Vorhandenes Verzeichnis auswählen (Ankopplung an Ethernet)

Im Fenster „Browse for Folder“ kann ein bestehendes Verzeichnis ausgewählt, in das Feld Folder übernommen und geändert werden.

- Mit OK erfolgt die Rückkehr in das Fenster „Select Installation Folder“ mit der Übernahme des Verzeichnisses.
- Mit Cancel erfolgt die Rückkehr. Dabei wird die Auswahl verworfen.

5. Die Installation der Software MOBY API ausführen

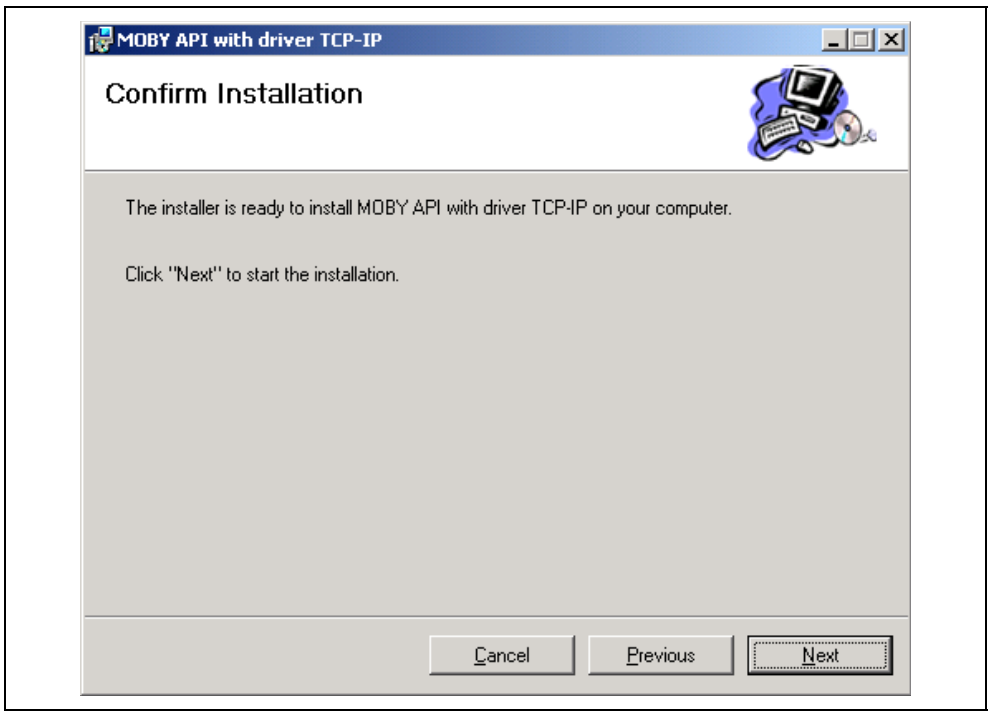


Bild 2-18 Installation ausführen (Ankopplung an Ethernet)

Mit „Next“ wird die Software MOBY API für die Erstellung von Applikationen unter den nachfolgend aufgeführten Unterverzeichnissen innerhalb des ausgewählten Verzeichnisses abgelegt:

\example	EXAMPLE.ZIP	Beispielapplikationen
\include	MOBY_API_T.H	Headerdatei für Anwenderapplikationen
\lib	MOBY_API_T.LIB	C-Bibliothek für Anwenderapplikationen

Die DLL von der dynamischen Link-Bibliothek MOBY_API_T wird in Abhängigkeit des Windows-Systems unter folgendes Verzeichnis kopiert:

Windows NT 4.0:	C:\WINNT\SYSTEM32
Windows 98:	C:\WIN98\SYSTEM
Windows 2000:	C:\WIN2000\SYSTEM32
Windows XP:	C:\WINNT\SYSTEM32

6. Vorgang „Bestehende C-Bibliothek MOBY API installieren“ ist beendet

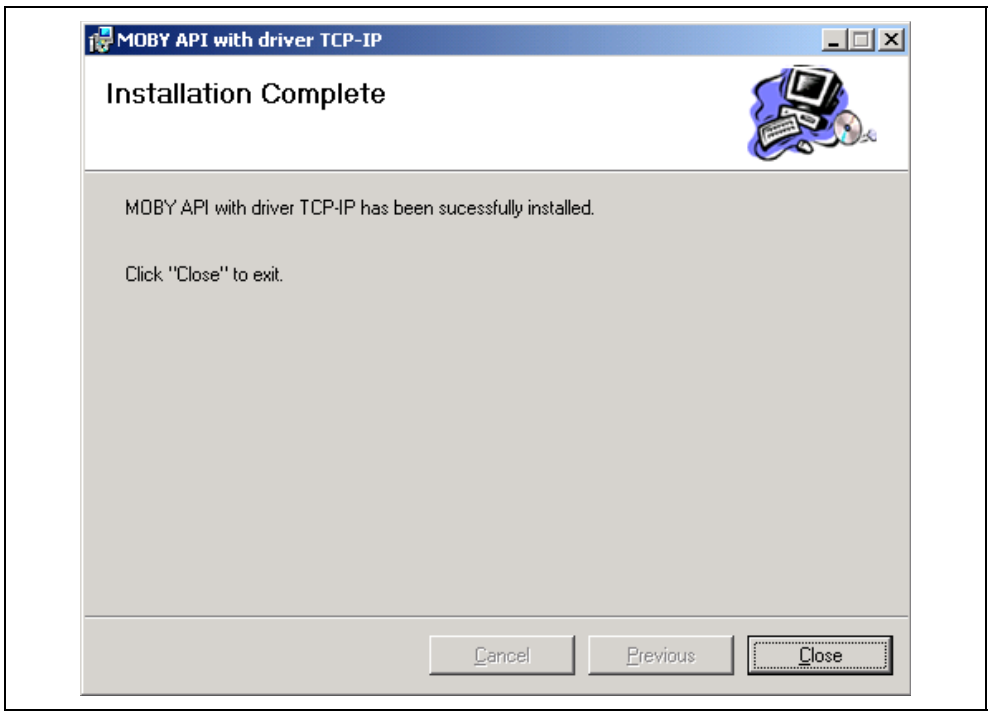


Bild 2-19 Installation vollständig (Ankopplung an Ethernet)

Mit Close wird der Dialog beendet.

Bestehende C-Bibliothek überschreiben

7. Bestehende C-Bibliothek MOBY API überschreiben oder deinstallieren

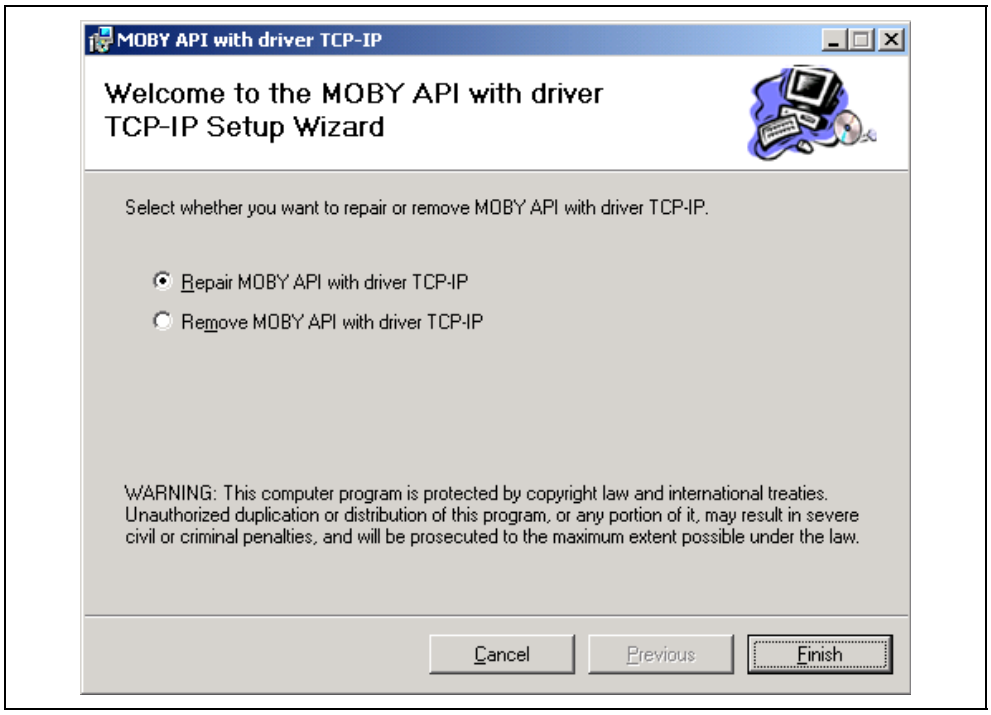


Bild 2-20 Bestehende C-Bibliothek überschreiben (Ankopplung an Ethernet)

Wenn die bestehende C-Bibliothek MOBY API überschrieben werden soll, ist „Repair MOBY API with driver TCP/IP“ zu selektieren und anschließend mit Finish der Vorgang zu starten (siehe Bild 2-21).

Achtung

Achten Sie darauf, dass Sie die neue Headerdatei MOBY_API_T.H und die C-Library MOBY_API_T.LIB in das Entwicklungsprojekt übernehmen.

8. Bestehende C-Bibliothek MOBY API überschreiben

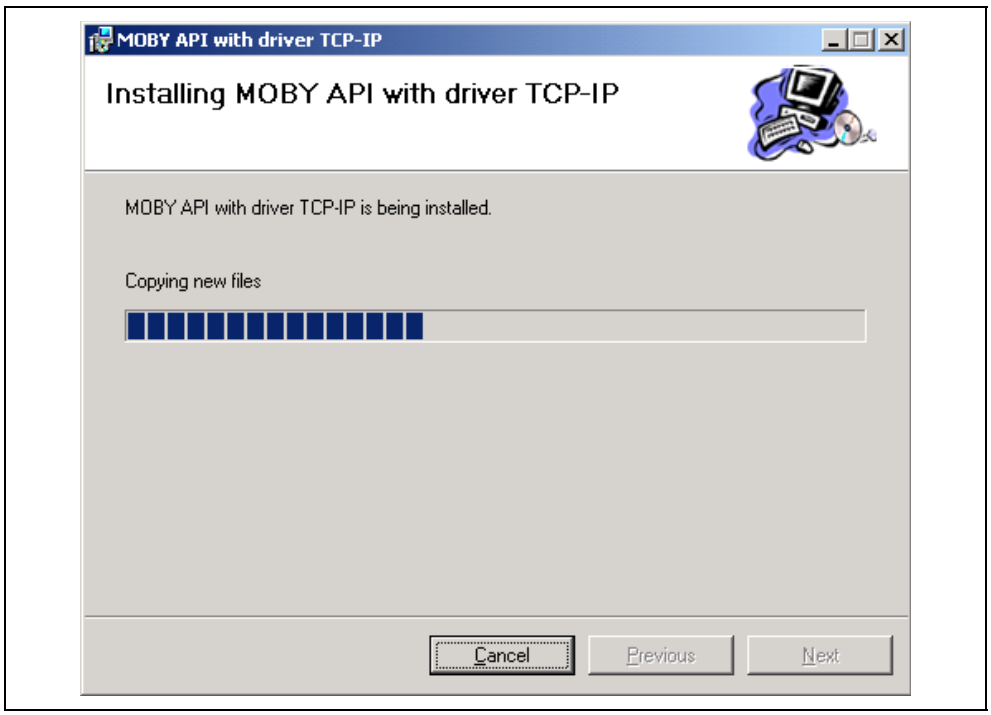


Bild 2-21 Fenster während des Installationsvorgangs „Überschreiben“ (Ankopplung an Ethernet)

Die bestehende C-Bibliothek MOBY API wird überschrieben.

9. Vorgang „Bestehende C-Bibliothek MOBY API überschreiben“ ist beendet

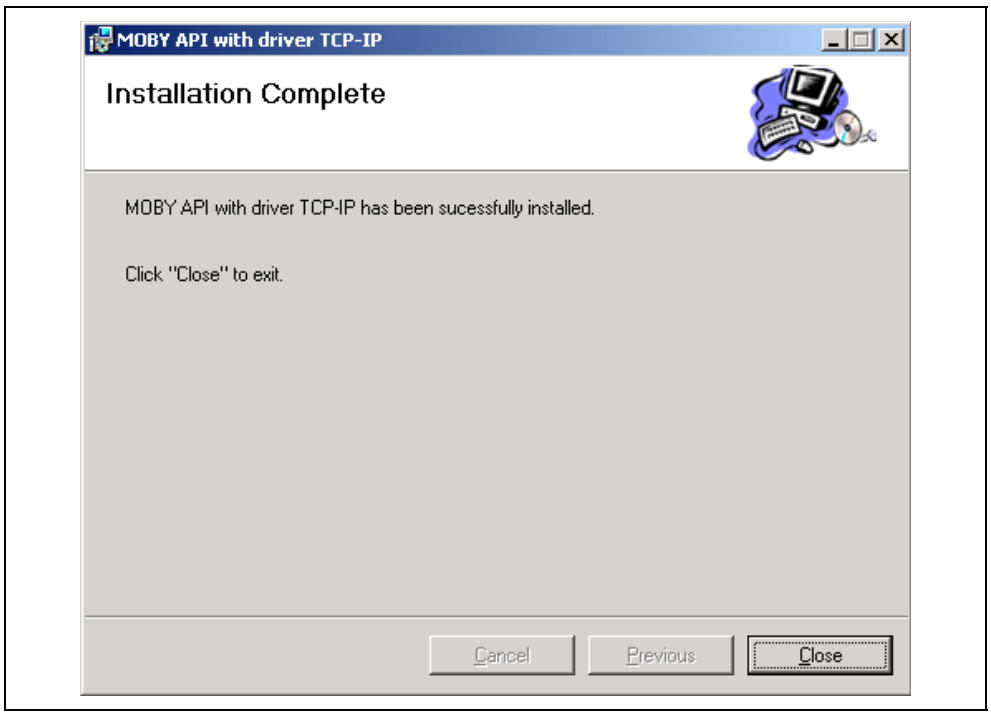


Bild 2-22 „Überschreiben“ ist vollständig (Ankopplung an Ethernet)

Mit Close wird der Dialog beendet.

Deinstallation

Wenn die bestehende C-Bibliothek MOBY API deinstalliert werden soll, ist „Remove MOBY API with driver TCP/IP“ zu selektieren (siehe Bild 2-20) und anschließend mit Finish der Vorgang zu starten (siehe Bild 2-23).

10. Bestehende C-Bibliothek MOBY API deinstallieren

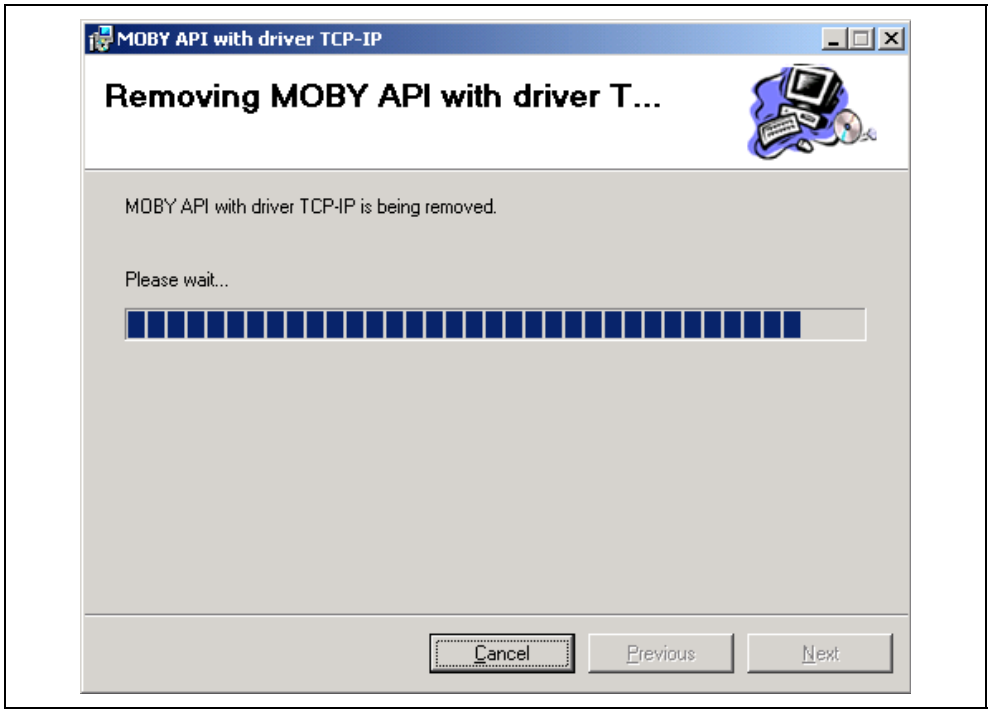


Bild 2-23 Fenster während der Deinstallation (Ankopplung an Ethernet)

11. Vorgang „Bestehende C-Bibliothek MOBY API deinstallieren“ ist beendet

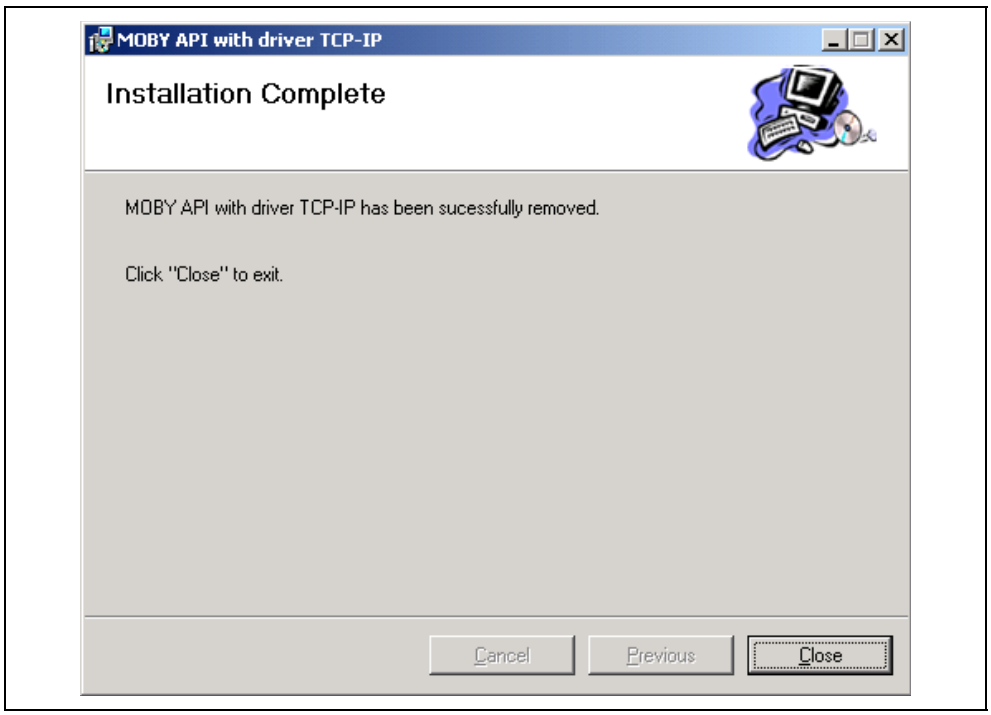


Bild 2-24 Deinstallation ist vollständig (Ankopplung an Ethernet)

Mit Close wird der Dialog beendet.

2.3.1 ASM 480 parametrieren

Das SLG wird über das Anschaltmodul ASM 480 mit dem Ethernet verbunden. Die Kommunikation zwischen der Applikation im PC (Client) und dem SLG (über das ASM 480 als Server) funktioniert nur mit einer eindeutigen Adresszuordnung:

- Die physikalische Adresse, **MAC-ID** (Media Access Control Identity), wird für jedes ASM 480 vom Hersteller festgelegt.
- Darüber hinaus benötigt jedes ASM 480 eine logische Adresse, **IP-Adresse** (Internet Protokoll), über die es im Netzwerk adressiert wird.

Die IP-Adresse darf im Netzwerk nur einmal vorhanden sein. Sie muss im ASM 480 parametriert werden. In der Anwenderapplikation im PC wird die IP-Adresse beim Verbindungsaufbau in der Funktion moby_open als Zeichenstring angegeben.

Die IP-Adresse besteht immer aus 32 Bit und wird im Dezimalformat (Wertebereich von 0 bis 255) dargestellt. Sie ist somit ein Zeichenstring aus vier Zahlenwerten in ASCII-Format, jeweils durch einen Punkt getrennt.

Zur Bestimmung des Netzwerks wird die **Subnetzmaske** (Subnetmask) benötigt. Die Subnetzmaske ist der IP-Adresse ähnlich. Sie besteht aus vier durch einen Punkt getrennten Zahlen (Wertebereich von 0 bis 255).

Beispiel: IP-Adresse „157.163.170.12“; Subnetzmaske „255.255.0.0“

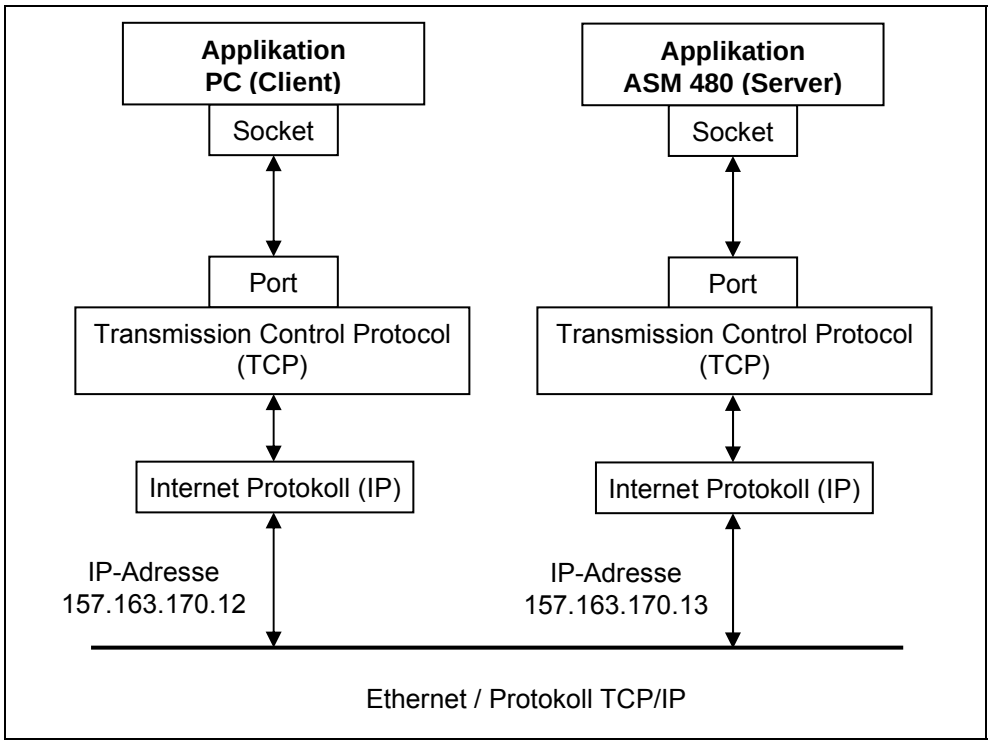


Bild 2-25 TCP/IP-Verbindung

Wenn die Kommunikation netzübergreifend zwischen verschiedenen Netzadressen erfolgt, so muss im ASM 480 noch die **IP-Adresse vom Standard-Gateway** parametriert werden. Die IP-Funktionalität bleibt ohne Angabe dieser Adresse auf das lokale Subnetz beschränkt.

TCP/IP-Konfiguration: IP-Adresse, Subnetzmaske und IP-Adresse vom Standard-Gateway am ASM 480 einstellen

Die Einstellung der TCP/IP-Konfiguration erfolgt direkt am ASM 480 mit Hilfe

- der vier Pfeiltasten: hoch, tief, rechts, links
- der beiden Menütasten: ESC, OK
- des LC-Displays

Für die Konfiguration muss das ASM 480 mit 24 V versorgt werden.

1. Einstellen der IP-Adresse

a)	Taste OK drücken:	Anzeige: „Menu; Trace“
b)	1x Pfeiltaste ‚tief’ drücken:	Anzeige: „Menu; Parameter“
c)	Taste OK drücken:	Anzeige: „Parameter; 3964R.1“
d)	2x Pfeiltaste ‚tief’ drücken:	Anzeige: „IP-Adress 157.163.170.022“ (z.B.)
e)	Taste OK drücken:	Anzeige: „IP Addr Byte0:; xxx“
f)	Taste OK drücken:	Mit den Pfeiltasten ‚hoch/tief’ die Ziffer, mit den Pfeiltasten ‚rechts/links’ die Dezimalstelle des jeweiligen Adressteils eingeben. Mit der Taste OK die Eingabe bestätigen.

2. Einstellen der Subnetzmaske

Im Anschluss an die Einstellung der IP-Adresse erfolgt die Eingabe der 4 Byte der Subnetzmaske (Net Mask:) (wie unter f) bei Einstellen der IP-Adresse beschrieben).

3. Einstellen Standard-Gateway-Adresse

Im Anschluss an die Einstellung der Subnetzmaske erfolgt die Eingabe der 4 Byte Gateway-Adress (Gateway:) (wie unter f) bei Einstellen der IP-Adresse beschrieben).

Nach allen Eingaben wird mit der Taste ESC in das Hauptmenü zurückgekehrt. Durch erneutes Drücken der Taste ESC wird der Eingabemodus verlassen. Die durchgeführten Änderungen werden gespeichert, wenn jetzt auf die Frage „Save Parameter and Restart?“ die Taste OK gedrückt wird.

Serielle Schnittstelle am ASM 480 einstellen

Am ASM 480 können die SLG U92-Varianten mit RS 232- oder RS 422-Schnittstelle betrieben werden.
In Abhängigkeit der SLG U92-Variante ist der serielle Schnittstellentyp am ASM 480 zu parametrieren.

- Menü „Parameter/3964R.1": Parameter „RTS-Control": **RS232 RTS off** oder **RS422 RTS off**

Das ASM 480 ist standardmäßig auf den Schnittstellentyp RS 232 (RS232 RTS OFF) eingestellt.

Achtung

Folgende Parameter dürfen im ASM 480 nicht verändert werden.

• Menü „Parameter/TCP":	Parameter „CR0 Socket Type"	„Server"
• Menü „Parameter/3964R.1":	Parameter „BaudRate"	38400 (oder 19200)
• Menü „Parameter/3964R.1":	Parameter „DatenBits"	8
• Menü „Parameter/3964R.1":	Parameter „StopBits"	1
• Menü „Parameter/3964R.1":	Parameter "Parity"	odd

Weitere Parameter für das Protokoll 3964R (Menü „Parameter/3964R.1":) sind:

- Priority 3964R high
- Idle Time [ms] 0
- ReceiveMode byte telegram
- Sendemode word tel MSB/LSB
- Error-SCC set/clear
- Send Repetition 5
- Receipt Delay 150
- Character Delay 50

Achtung

Der Parameter „CR0 Port Number" ist mit dem Default-Wert 8000 parametriert und muss bei der Funktion moby_open unter Parameter sPort (siehe Kapitel 3.2.2) angegeben werden.

- Menü „Parameter/TCP": Parameter „CR0 Port Number" 8000

3 C-Bibliothek MOBY API

Für das Betreiben der SLG oder SLA an ASM und SIM stehen eine Reihe von Funktionen in den C-Bibliotheken

- MOBY_API.LIB (für die serielle Ankopplung an PC)
- MOBY_API_T.LIB (für die Ankopplung an Ethernet)

zur Verfügung, mit denen die gesamte Kommunikation abgewickelt werden kann.

Die C-Bibliotheksfunktionen sind unterteilt in vier Funktionsgruppen:

- Schnittstellenfunktionen
- Systemfunktionen
- MDS-Funktionen
- DE-/DA-Funktionen

Darüber hinaus steht für die Abfrage der Version der dynamischen Link-Bibliothek MOBY_API.DLL oder MOBY_API_T.DLL die Funktion moby_version zur Verfügung.

In der nachfolgenden Tabelle sind die Funktionen mit der Zuordnung zu SLG/ASM/SIM aufgeführt.

Tabelle 3-1 Zuordnung der Bibliotheksfunktionen zu SLG/ASM/SIM

Funktion	ASM 724 mit SLA 71	ASM 824 mit SLA 81	ASM 424 mit ...		ASM 420 mit ...		SIM 41	SLG U92	ASM 480 mit SLG U92
			SLG 7x	SLG 4x	SLG 7x	SLG 4x			
Schnittstellenfunktionen									
moby_open	x	x	x	x	x	x	x	x	x
moby_close	x	x	x	x	x	x	x	x	x
Systemfunktionen									
moby_start	x	x	x	x	x	x	x	x	x
moby_stop	x	x	x	x	x	x	x	x	x
moby_next	x	–	x	x	x	x	x	–	–
moby_end	–	–	–	–	–	–	–	x	–
moby_s_end	–	–	–	–	–	–	–	x	x
moby_setANT	–	–	–	–	–	–	–	x	x
moby_repeat	–	–	–	–	–	–	–	x [1]	x [1]
moby_anw	–	–	–	–	–	–	–	x	x
moby_status	x	x	x	x	x	x	x	–	–
moby_statusU	–	–	–	–	–	–	–	x	x
moby_diagnose	–	–	–	–	–	–	–	x	x
moby_unexpect	x	x	x	x	x	x	x	x	x

1 In Vorbereitung

Tabelle 3-1 Zuordnung der Bibliotheksfunktionen zu SLG/ASM/SIM

Funktion	ASM 724 mit SLA 71	ASM 824 mit SLA 81	ASM 424 mit ...		ASM 420 mit ...		SIM 41	SLG U92	ASM 480 mit SLG U92
			SLG 7x	SLG 4x	SLG 7x	SLG 4x			
MDS-Funktionen									
moby_read	x	x	x	x	x	x	x	x	–
moby_getID	x	x	x	–	x	–	–	x	–
moby_write	x	x	x	x	x	x	x	x	–
moby_init	x	x	x	x	x	x	x	x	–
moby_statusMDS	–	–	–	–	–	–	–	x	–
moby_readOTP	–	–	–	–	–	–	–	x	–
moby_writeOTP	–	–	–	–	–	–	–	x	–
moby_s_read	–	–	–	–	–	–	–	x	x
moby_s_getID	–	–	–	–	–	–	–	x	x
moby_s_write	–	–	–	–	–	–	–	x	x
moby_s_init	–	–	–	–	–	–	–	x	x
moby_s_copy	–	–	–	–	–	–	–	x	x
moby_s_statusMDS	–	–	–	–	–	–	–	x	x
moby_s_readOTP	–	–	–	–	–	–	–	x	x
moby_s_writeOTP	–	–	–	–	–	–	–	x	x
DE-/DA-Funktionen									
moby_readDE	–	–	–	–	x	x	x	–	–
moby_writeDA	–	–	–	–	x	x	x	–	–
Versionsabfrage									
moby_version	x	x	x	x	x	x	x	x	x

Die System-, die MDS- und die DE-/DA-Funktionen erzeugen Telegramme an das SLG, ASM oder SIM und erwarten immer ein Antworttelegramm vom SLG, ASM oder SIM. Sie warten jedoch nicht direkt auf das Antworttelegramm. Sie werden wieder fortgesetzt, wenn das Telegramm erfolgreich ohne Fehler abgeschickt oder mit Fehler abgebrochen wurde. Die Applikation muss anschließend auf das Antworttelegramm über ein initialisiertes Windows-Event warten, das beim Funktionsaufruf mit zu übergeben ist. Sobald das Antworttelegramm empfangen wurde, wird das entsprechende Event gesetzt. Der Applikation wird damit signalisiert, dass die aufgerufene Funktion vollständig abgeschlossen ist.
Durch diese Vorgehensweise sind alle Funktionen, die sich auf SLG, SIM oder ASM auswirken, nicht blockierend.

Achtung

- Bei ASM 424, ASM 420 mit SLG 4x und SIM 41 sind die MDS-Funktionen mit ECC-Korrektur möglich.
- Das Windows-Event muss mit CreateEvent erzeugt werden.

3.1 Generelles zur Benutzung der C-Bibliothek MOBY API

3.1.1 Synchronisation

Jede Routine, die nach vollständiger Abwicklung des Befehls ein Antworttelegramm des SLG, ASM oder SIM (MOBY-Device) an den Host auslöst, muss synchronisiert werden. Um dies zu ermöglichen, hat jede von diesen Routinen einen Parameter vom Typ „HANDLE“. Dieser Parameter muss auf ein initialisiertes Windows-Event verweisen, das mit CreateEvent erzeugt wurde.
Beim Aufruf einer derartigen Routine wird der aufrufende Thread so lange blockiert, bis die entsprechende Aktion gestartet ist, das heißt, bis ein eventuelles Telegramm an das SLG, ASM oder SIM vollständig abgeschickt ist. Danach kehrt die aufgerufene Routine zurück, ohne auf ein Antworttelegramm zu warten.
Sobald das entsprechende Antworttelegramm vom SLG, ASM oder SIM empfangen wird, wird das entsprechende Event gesetzt und damit dem aufrufenden Thread signalisiert, dass die aufgerufene Funktion nun vollständig abgeschlossen ist. Eventuelle Fehlerinformationen im Antworttelegramm werden in ein Speicherwort geschrieben, das beim Aufruf der Routine spezifiziert wurde.

3.1.2 Antworttelegramm-Matching

Um die oben beschriebene Funktion zu ermöglichen, müssen eingehende Telegramme mit vorangegangenen Kommandotelegrammen abgeglichen – gematched – werden. Dies geschieht auf Basis der Kommandonummern und bei gleicher Kommandonummer nach dem FIFO-Prinzip.

3.1.3 Unerwartete Telegramme

Es kann vorkommen, dass unerwartete Telegramme vom SLG, ASM oder SIM empfangen werden. Unerwartete Telegramme können sein

- Hochlaufmeldung „02 00 0F“ hex
- Telegramme nach zurückgesetzter MOBY-DLL, die noch vom SLG, ASM oder SIM zurückkommen, deren Eintrag im Matchpuffer aber gelöscht wurde.

Um diese zu behandeln, wird der aufrufende Prozess benachrichtigt. Dazu kann eine Callback-Routine spezifiziert werden, die beim Eintreffen eines unerwarteten (nicht gematchten) Telegramms aufgerufen wird.
Es besteht auch die Möglichkeit, unerwartete Telegramme zu ignorieren (siehe Kapitel 3.3.12).

3.1.4 Verbindungsüberwachung

Bei der Ankopplung an Ethernet wird die Verbindung zwischen der MOBY-DLL und dem einzelnen SLG auf Telegrammebene überwacht. Wenn der Zeitabstand nach dem letzten Telegramm vom entsprechenden SLG größer als 3 Sekunden wird, dann sendet die MOBY-DLL automatisch ein Überwachungstelegramm an das SLG und erwartet innerhalb von 10 Sekunden eine Rückmeldung vom SLG. Wenn nicht innerhalb dieser 10 Sekunden ein Telegramm vom SLG kommt, so wird die Verbindung zum SLG als unterbrochen angesehen. Die MOBY-DLL schließt die Verbindung. Ein anstehendes Telegramm wird mit Fehler abgebrochen. Wenn kein Telegramm ansteht, kommt der Fehler erst beim nächsten Telegramm. Die Applikation muss vor einer erneuten Kommunikation die Verbindung wieder mit moby_open und moby_start öffnen.

3.1.5 Fehlerfall

Es kann vorkommen, dass Telegramme vom SLG, ASM oder SIM fehlerhaft bzw. nicht vollständig empfangen werden. In diesem Fall wird implizit „moby_stop“ aufgerufen, das heißt, die Benutzung der Schnittstelle wird blockiert und alle anstehenden Aufträge werden mit entsprechender Fehlermeldung abgebrochen. Um die Schnittstelle wieder benutzen zu können, muss erneut „moby_start“ und damit ein RESET des SLG, ASM oder SIM ausgeführt werden.

Achtung

Für den Fall, dass ein eventueller Fehler nicht eindeutig einer geöffneten Schnittstelle zugeordnet werden kann, z. B. bei einem Hardwarefehler auf einem ASM 824/724/424-Kanal, werden alle potentiell betroffenen Schnittstellen blockiert.

Achtung

Wenn Windows während des Sendevorgangs die Überwachungszeiten überschreitet, so kann es je nach Unterbrechungsstelle im Telegramm zu einem unerwarteten Telegramm kommen. Dieses Telegramm wird von der Callback-Routine gemeldet, wenn Sie die Funktion moby_unexpect aufgerufen haben. Das Telegramm ist zu ignorieren.

3.1.6 Öffnen, Verwenden und Schließen von Schnittstellen

Bevor ein SLG, ASM oder SIM (MOBY-Device) mit Hilfe der MOBY API angesprochen werden kann, muss die entsprechende Schnittstelle geöffnet und initialisiert werden. Dies geschieht durch die Befehle „moby_open“ und „moby_start“. Anschließend kann auf die Schnittstelle mit beliebigen Lese- und Schreibbefehlen zugegriffen werden. Vor dem Programmende hat der Programmierer die Schnittstelle mit „moby_stop“ zu deaktivieren und anschließend mit „moby_close“ zu schließen. Dieser Vorgang ist grafisch im Bild 3-1 dargestellt.

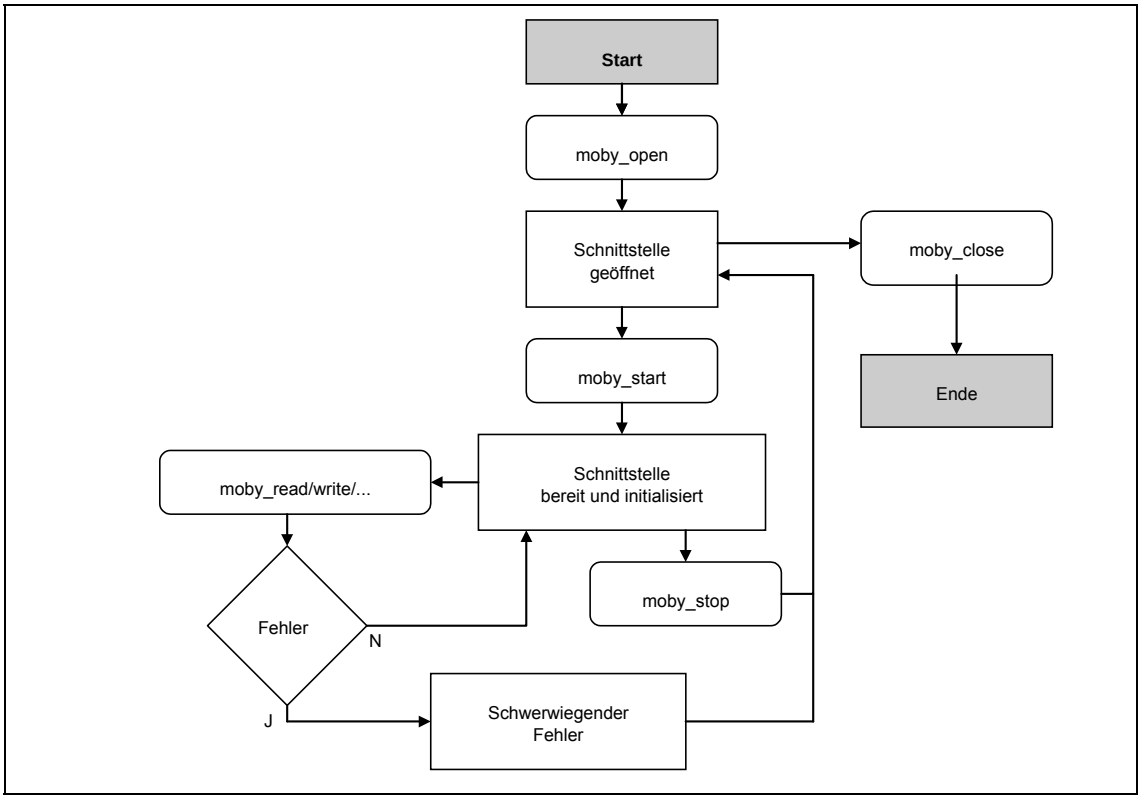


Bild 3-1 Funktionsfolge bei Nutzung der MOPY API

Das Bild 3-1 zeigt auch das Verfahren im Falle eines schweren Kommunikationsfehlers oder eines Fehlers von der Leitungsüberwachung. Dies führt zu einem automatischen „moby_stop“ und führt die Schnittstelle in einen gesicherten, aber nicht aktiven Zustand zurück.

Achtung

Bei einem externen Abbruch einer Applikation (z. B. über CRTL-ALT-DEL), die die C-Bibliothek MOBY API verwendet und mindestens einen Kanal geöffnet hat, kann es unter Windows 95/98 unter Umständen dazu kommen, dass Systemressourcen nicht korrekt freigegeben werden. Dadurch kann es bei der Ausführung nachfolgender MOBY API-Programme zu Problemen kommen. Sollte dies der Fall sein, ist das System neu zu starten.

3.1.7 Verwendung mehrerer Kommunikationskanäle

Die C-Bibliothek MOBY API ermöglicht die Verwendung mehrerer Kommunikationskanäle auf einer physikalischen Schnittstelle. Dies kann durch wiederholtes Anwenden des Befehls „moby_open“ erreicht werden. Dabei ist zu beachten, dass alle Kommunikationskanäle einer physikalischen Schnittstelle vom selben Prozess (von der selben Applikation) zu öffnen sind.

3.1.8 Logdatei(en) ERRx.TXT

Bei der Anwendung mit TCP/IP erzeugt das System für jede Verbindung (Kommunikationskanal) eine Logdatei „ERRx.TXT“ mit maximal 7 kByte Länge. x steht für die Kanalnummer.
Die Logdatei(en) wird(werden) unter dem Pfad abgelegt, von dem aus die Applikation aufgerufen wird. Die Logdatei beinhaltet systeminterne Diagnosedaten.

3.2 Schnittstellenfunktionen

Die Schnittstellenfunktionen moby_open und moby_close öffnen und schließen den Kommunikationskanal für die Kommunikation über

- die serielle Schnittstelle oder
- die Ethernet-Schnittstelle

Die Ethernet-Schnittstelle kann zur Zeit nur für MOBY U mit dem SLG U92 als MOBY-Typ „MOBY_Uc“ (siehe Kapitel 3.3.1) mit MOBY U-Befehlen für Singletag- oder Multitag-Betrieb genutzt werden.

3.2.1 Funktion moby_open für die serielle Schnittstelle

mobyErr_t moby_open (char *comStr, int channel, mobyHandle_t *handle);

Diese Funktion öffnet das angegebene MOBY-Device mit Hilfe des Treibers. Der Parameter „handle“ wird bei erfolgreichem Befehlsabschluss dazu verwendet, ein Handle auf das geöffnete MOBY-Device zurückzugeben. Es muss bei allen folgenden Aufrufen, die auf dieses Device zugreifen, übergeben werden. Zuvor muss ein geöffnetes MOBY-Device jedoch mit „moby_start“ parametriert und initialisiert werden.

Parameter	Typ	Beschreibung
comStr	char *	Zeichenkette mit dem Namen der seriellen Schnittstelle. „COM1“, „COM2“, „COM3“ oder „COM4“
channel	int	Kanalnummer, unter der das SLG, das SIM oder die SLA angesprochen wird. 1 bis 4 für SLA oder SLG 4x an ASM 424 oder 0, wenn SLG U92, SLG xx an ASM 420 oder SIM 4x.
handle	mobyHandle_t *	Pointer auf das Handle der seriellen Schnittstelle und der Kanalnummer (Rückgabewert nach fehlerfreiem Funktionsdurchlauf). Das zurückgegebene Handle wird für alle anderen Funktionen als Eingabeparameter zur Identifikation der seriellen Schnittstelle und der Kanalnummer verwendet.

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Windows kann die serielle Schnittstelle nicht öffnen (siehe Kap. 3.7.1).

Achtung

Die Kanalnummer wird automatisch bei jeder Funktion, die sich auf diese Schnittstelle bezieht, hinzugefügt.

3.2.2 Funktion moby_open für die Ethernet-Schnittstelle

mobyErr_t moby_open (char *comStr, int channel, mobyHandle_t *handle, unsigned short sPort);

Diese Funktion öffnet das angegebene MOBY-Device mit Hilfe des Treibers. Der Parameter „handle“ wird bei erfolgreichem Befehlsabschluss dazu verwendet, ein Handle auf das geöffnete MOBY-Device zurückzugeben. Es muss bei allen folgenden Aufrufen, die auf dieses Device zugreifen, übergeben werden. Zuvor muss ein geöffnetes MOBY-Device jedoch mit „moby_start“ parametriert und initialisiert werden.

Parameter	Typ	Beschreibung
comStr	char *	IP-Adresse des ASM 480 als Zeichenstring (siehe Kapitel 2.3.1) z. B: „157.163.170.12“
channel	int	Kanalnummer, unter der das SLG angesprochen wird. 0 für SLG U92
handle	mobyHandle_t *	Pointer auf das Handle der Ethernet-Schnittstelle (Rückgabewert nach fehlerfreiem Funktionsdurchlauf). Das zurückgegebene Handle wird für alle anderen Funktionen als Eingabeparameter zur Identifikation der Ethernet-Schnittstelle verwendet.
sPort	unsigned short	Portnummer, unter der das ASM 480 angesprochen wird (siehe Kapitel 2.3.1).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Windows kann die Ethernet-Schnittstelle nicht öffnen (siehe Kap. 3.7.1).

3.2.3 Funktion moby_close

mobyErr_t moby_close (mobyHandle_t handle);

Diese Funktion schließt mit Hilfe des Treibers das angegebene MOBY-Device. Anschließend kann auf das MOBY-Device nicht mehr zugegriffen werden. Diese Routine muss durchlaufen werden, wenn die Kommunikation mit dem MOBY-Device mit „moby_start“ (siehe Kap. 3.3.1) aufgenommen wurde. Sie wird allerdings nur dann ausgeführt, wenn der Benutzer zuvor „moby_stop“ (siehe Kapitel 3.3.2) aufgerufen hat; anderfalls wird ein Fehler zurückgeliefert.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer oder der Ethernet-Schnittstelle.

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Windows kann die serielle Schnittstelle oder die Ethernet-Schnittstelle nicht schließen (siehe Kapitel 3.7.1).

3.3 Systemfunktionen

Mit den Systemfunktionen steuern Sie die Kommunikation mit einem SLG oder einer SLA an einem ASM oder einem SIM. Sie stellen den Funktionsrahmen für die MDS-Funktionen (siehe Kapitel 3.4) und wirken nicht direkt auf die MDS.

Tabelle 3-2 Zuordnung der Systemfunktionen zu den Schnittstellen

Funktion	Serielle Schnittstelle	Ethernet-Schnittstelle
moby_start	X	X
moby_stop	X	X
moby_next	X	–
moby_end	X	–
moby_s_end	X	X
moby_setANT	X	X
moby_repeat	X	X
moby_status	X	–
moby_statusU	X	X
moby_anw	X	X
moby_diagnose	X	X
moby_unexpect	X	X

3.3.1 Funktion moby_start

mobyErr_t moby_start (mobyHandle_t handle, mobyType_t type, BOOL eccON, mobyParameters_t *param, HANDLE sync, mobyErr_t *err);

Diese Funktion parametriert und initialisiert ein bereits geöffnetes MOBY-Device. Dabei wird unter anderem auch ein RESET-Telegramm, wahlweise mit oder ohne Parameterübergabe, an das SLG, ASM oder SIM geschickt. Erst nach Abschluss dieses Telegrammes, erfolgreich oder nicht, ist die Schnittstelle bereit für weitere Telegramme.
Wenn diese Funktion nach erfolgreicher Durchführung erneut aufgerufen werden soll, so muss zuerst die Funktion moby_stop ausgeführt werden.
Die Parameter für die RESET-Telegramme werden mit Hilfe einer Union-Datenstruktur (siehe Headerdatei MOBY_API.H, Kapitel 4.1 bzw. Headerdatei MOBY_API_T.H, Kapitel 4.3) übergeben. Die Union-Datenstruktur wird in Abhängigkeit von einer Konstanten ausgewertet, über die der MOBY-Typ (-System) vorgegeben wird.

Tabelle 3-3 Konstanten für den MOBY-Typ

Konstante (siehe Parameter type)	MOBY-Typ	Schnittstelle		Kommentar
		Seriell	Ethernet	
MOBY_E	SLG 7x an ASM 420 oder SLA 7x an ASM 724	X	–	
MOBY_F	SLA 8x an ASM 824	X	–	
MOBY_I oder MOBY_Ia	SIM 41, SLG 4x an ASM 420 oder SLG 4x an ASM 424	X	–	Bei Verwendung der Option mit Parameter wird standardmäßig das RESET-Telegramm für das Setzen der Intervallparameter verwendet.
MOBY_Ib	SIM 41, SLG 4x an ASM 420 oder SLG 4x an ASM 424	X	–	Bei der Angabe von „MOBY_Ib“ wird die zweite Parametervariante, die Übergabe von erweiterten Parametern (wie z. B. LED Settings), angewendet.
MOBY_Ua	SLG U92	X	–	Kurzes RESET-Telegramm (MOBY I-aufrufkompatibel, kein Multitag-Betrieb)
MOBY_Ub	SLG U92	X	–	Langes RESET-Telegramm (MOBY I-aufrufkompatibel, kein Multitag-Betrieb)
MOBY_Uc	SLG U92	X	–	Langes RESET-Telegramm (MOBY U-Befehle für Singletag- oder Multitag-Betrieb) • Wert in param = 1 (Singletag-Betrieb) • Wert in param > 1 (Multitag-Betrieb)
	SLG U92 an ASM 480	–	X	

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer oder der Ethernet-Schnittstelle.
type	mobyType_t	Zeichenkette mit dem Namen des MOBY-Systems. MOBY_E MOBY_F MOBY_I MOBY_Ia MOBY_Ib MOBY_Ua MOBY_Ub MOBY_Uc
eccON	BOOL	Boolscher Wert für MDS-Funktionen ohne oder mit ECC-Korrektur. FALSE = ohne ECC-Korrektur TRUE = mit ECC-Korrektur
param	mobyParameters_t *	Pointer für Parameterübergabe NULL = keine Parameterübergabe (Standard-RESET) oder Pointer auf den Puffer, in dem die zu übergebenden Parameter (RESET-Parameter) abgelegt sind.
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Die RESET-Parameter sind MOBY-spezifisch und der entsprechenden MOBY-Dokumentation zu entnehmen.

- SLG U92 Siehe Anhänge B.2.1.2.1, B.3.1.2.1 und B.4.1.2.1
- SLG U92 an ASM 480 Siehe Anhang B.4.1.2.1
- SLG an ASM 420 Siehe /01/
- SIM 41 Siehe /02/
- SLA 8x an ASM 824 Siehe Anhang A.6.2
- SLA 7x an ASM 724 Siehe Anhang A.6.2
- SLG 4x an ASM 424 Siehe Anhang A.6.2

Die Funktion moby_start entspricht bei MOBY E und I dem Telegramm RESET mit oder ohne Parameterübergabe.

3.3.2 Funktion moby_stop

mobyErr_t moby_stop (mobyHandle_t handle);

Diese Funktion setzt die Kommunikation mit einem SLG oder SLA an einem ASM oder SIM und der MOBY-DLL zurück. Nach der Ausführung können vor erneutem „moby_start“ keine weiteren Befehle an das Device abgesetzt werden. Noch anstehende Befehle werden abgebrochen und als solche an wartende Threads signalisiert.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer oder der Ethernet-Schnittstelle.

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Diese Routine setzt kein Telegramm an das SLG, ASM oder das SIM ab.

3.3.3 Funktion moby_next

mobyErr_t moby_next (mobyHandle_t handle, HANDLE sync, mobyErr_t *err);

Wenn der letzte bearbeitete MDS sich noch im Feld befindet, dann können Sie nach dieser Funktion schon die Bearbeitung des nächsten MDS aktivieren.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer.
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

3.3.4 Funktion moby_end

mobyErr_t moby_end (mobyHandle_t handle, unsigned char mode, HANDLE sync, mobyErr_t *err);

Mit dieser Funktion wird für den zuletzt bearbeiteten und noch im Feld des SLG befindlichen MDS die Standby-Zeit (im RESET-Telegramm parametriert) unwirksam geschaltet, um den Stromverbrauch des MDS zu reduzieren.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer.
mode	unsigned char	00 hex = Die Bearbeitung mit dem MDS ist beendet und die parametrierte Standby-Zeit soll unwirksam werden. Mit diesem MDS soll keine Kommunikation mehr stattfinden. Der MDS wird das Feld des SLG verlassen. Das SLG trägt den MDS aus der Bearbeitungsliste aus und führt den MDS weiterhin in der Anwesenheitsliste bis der MDS das Feld des SLG verlassen hat. 01 hex = Es besteht eine Bearbeitungspause mit dem MDS und die parametrierte Standby-Zeit soll unwirksam werden. Der MDS verlässt noch nicht das Feld des SLG. Es ist noch mindestens eine weitere Kommunikation mit dem MDS vorgesehen. Das SLG führt den MDS weiterhin in der Bearbeitungsliste und in der Anwesenheitsliste.
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Der Befehl END darf nur nach dem Befehl moby_write, moby_read, moby_init oder moby_statusMDS an das SLG abgesetzt werden, und es darf kein Befehl beim SLG anstehen. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.
Wenn der MDS das Feld des SLG bereits verlassen hat und mode gleich 01 gewählt wurde, kommt eine Fehlermeldung. Ist dagegen ein weiterer MDS in das Feld des SLG eingetreten und mode gleich 01 gewählt, so kommt ebenso eine Fehlermeldung.

Achtung

Der Befehl kann nur bei MOBY U; Typ MOBY_Ua oder MOBY_Ub (siehe Tabelle 3-3 angewendet werden, d. h. MOBY U ohne Multitag-Betrieb.

3.3.5 Funktion moby_s_end

mobyErr_t moby_s_end (mobyHandle_t handle, uInt *idData, unsigned char mode, HANDLE sync, mobyErr_t *err);

Mit dieser Funktion schalten Sie „gezielt“ oder „ungezielt“ die Standby-Zeit (im RESET-Telegramm parametriert) eines MDS unwirksam, der sich im Antennenfeld des SLG U92 befindet.

- Es ist ein „gezielter“ Leseaufruf, wenn der Aufruf mit Identnummer des MDS abgesetzt wird. Mehrere MDS dürfen sich im Antennenfeld befinden. Die Identnummer des MDS können Sie über die Funktion GET ermitteln.
- Es ist ein „ungezielter“ Leseaufruf, wenn der Aufruf ohne Identnummer des MDS abgesetzt wird. Es darf sich nur ein MDS im Antennenfeld befinden.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer oder der Ethernet-Schnittstelle.
idData	uInt *	Wert von 2^0 bis 2^{32}-1 Identnummer des MDS 0 wenn sich nur ein MDS im Feld befindet und ein „ungezielter“ Leseaufruf erfolgen soll
mode	unsigned char	00 hex = Die Bearbeitung mit dem MDS ist beendet und die parametrierte Standby-Zeit soll unwirksam werden. Mit diesem MDS soll keine Kommunikation mehr stattfinden. Der MDS wird das Feld des SLG verlassen. Das SLG trägt den MDS aus der Bearbeitungsliste aus und führt den MDS weiterhin in der Anwesenheitsliste, bis der MDS das Feld des SLG verlassen hat. 01 hex = Es besteht eine Bearbeitungspause mit dem MDS und die parametrierte Standby-Zeit soll unwirksam werden. Der MDS verlässt noch nicht das Feld des SLG. Es ist noch mindestens eine weitere Kommunikation mit dem MDS vorgesehen. Das SLG führt den MDS weiterhin in der Bearbeitungsliste und in der Anwesenheitsliste.
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0	kein Fehler, Befehl ausgeführt
< 0	Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Der Befehl moby_s_end darf nur nach dem Befehl moby_s_write, moby_s_read, moby_s_init oder moby_s_statusMDS an das SLG abgesetzt werden, und es darf kein Befehl beim SLG anstehen. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.
Wenn sich beim „gezielten“ Aufruf der MDS mit der vorgegebenen Identnummer nicht in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.
Wenn der MDS das Feld des SLG bereits verlassen hat und mode = 01 gewählt wurde, kommt eine Fehlermeldung. Ist dagegen ein weiterer MDS in das Feld des SLG eingetreten und mode = 01 gewählt, so kommt ebenso eine Fehlermeldung.
Wenn sich beim „ungezielten“ Aufruf mehr als ein MDS in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.

3.3.6 Funktion moby_setANT

mobyErr_t moby_setANT (mobyHandle_t handle,
unsigned char mode, mobyErr_t *err);

Mit dieser Funktion wird die Antenne des Schreib-/Lesegerätes (SLG) ein- oder ausgeschaltet.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer oder der Ethernet-Schnittstelle.
mode	unsigned char	01 hex = Antenne einschalten 02 hex = Antenne ausschalten
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Der Befehl moby_setANT darf nur an das SLG abgesetzt werden, wenn noch kein Befehl beim SLG ansteht.
Zum Zeitpunkt Antenne einschalten darf bereits ein MDS im Feld des SLG vorhanden sein.
Wenn beim Abschalten sich ein MDS im Feld des SLG befindet, so wird bei Betrieb mit Anwesenheit dieser als abwesend gemeldet.

3.3.7 Funktion moby_repeat [1]

mobyErr_t moby_repeat (mobyHandle_t handle, unsigned char mode, HANDLE sync, mobyErr_t *err);

Mit dieser Funktion wird der letzte ausgeführte Befehl wiederholt.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer oder der Ethernet-Schnittstelle.
mode	unsigned char	00 hex = Letzten Befehl wiederholen bis moby_repeat mit mode gleich 01 kommt. 01 hex = Wiederholung beenden. Ein begonnener Befehl wird zu Ende bearbeitet.
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Der Befehl mit mode gleich Null darf nur dann abgesetzt werden, wenn ein Befehl zur Ausführung ansteht, wie z. B. moby_read.

1 In Vorbereitung

3.3.8 Funktion moby_status

mobyErr_t moby_status (mobyHandle_t handle, mobyStatus_t *status, HANDLE sync, mobyErr_t *err);

Mit dieser Funktion fragen Sie den Status eines SLG oder einer SLA an einem ASM oder SIM ab.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer.
status	mobyStatus_t *	Pointer auf die Struktur, in die die gelesene Statusinformation abzulegen ist.
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Es ist jeweils nur ein ausstehendes Statustelegramm erlaubt. Jede Statusanfrage, die zu einem Zeitpunkt abgesetzt wird, zu dem noch eine Statusanfrage unbeantwortet ist, wird mit einem entsprechenden Fehler zurückgewiesen.
Die Statusinformation ist MOBY-spezifisch. Die Daten sind erst gültig, wenn durch das Synchronisations-Handle der Empfang des Antworttelegramms signalisiert wird. Der Inhalt und die Bedeutung sind der entsprechenden MOBY-Dokumentation /01/ bis /05/ oder auch Anhang A.6.6 für ASM 424/724/824 zu entnehmen.

3.3.9 Funktion moby_statusU

mobyErr_t moby_statusU (mobyHandle_t handle,
mobyStatusU_t *status,
HANDLE sync, mobyErr_t *err);

Mit dieser Funktion fragen Sie den Status eines SLG U92 ab.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer oder der Ethernet-Schnittstelle.
status	mobyStatusU_t *	Pointer auf die Struktur, in die die Status- und Diagnoseinformation des Antworttelegramms abzulegen ist.
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Es ist jeweils nur ein ausstehendes Statustelegramm erlaubt. Jede Statusanfrage, die zu einem Zeitpunkt abgesetzt wird, zu dem noch eine Statusanfrage unbeantwortet ist, wird mit einem entsprechenden Fehler zurückgewiesen.
Die Statusinformation ist MOBY-spezifisch. Die Daten sind erst gültig, wenn durch das Synchronisations-Handle der Empfang des Antworttelegramms signalisiert wird. Der Inhalt und die Bedeutung sind dem Anhang B.2.2.2.2 zu entnehmen.

3.3.10 Funktion moby_anw

mobyErr_t moby_anw (mobyHandle_t handle, moby_AnwCallback_t cbroutine);

Mit dieser Funktion wird die Anwesenheit der MDS im Feld des SLG gemeldet. Die Anwesenheitsmeldung kommt asynchron vom SLG und wird als Callback-Routine aufgerufen.

Wenn im RESET-Telegramm „Betrieb mit Anwesenheit“ gesetzt wurde, dann schickt das SLG nach jeder Anwesenheitsänderung im Feld des SLG ein Telegramm mit der Anzahl der im Feld befindlichen MDS. Tritt gleichzeitig ein MDS aus dem Feld und ein anderer MDS in das Feld, so werden zwei Telegramme verschickt. Treten gleichzeitig mehrere MDS ins Feld, so bewirkt jeder MDS eine Anwesenheitsmeldung. Das Gleiche gilt für das Verlassen des Feldes. In der Headerdatei MOBY_API.H oder MOBY_API_T.H ist der Übergabewert, Anzahl der im Feld befindlichen MDS, mit „typedef void (CALLBACK *moby_AnwCallback_t)(unsigned char anwstatus);“ definiert (siehe Kapitel 4.1 oder Kapitel 4.3).

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer oder der Ethernet-Schnittstelle.
cbroutine	Moby_AnwCallback_t	Callback-Routine für die Anwesenheitsmeldung vom SLG: MDS ist gekommen oder MDS ist gegangen.

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Wenn die Funktion moby_anw nicht im Anwenderprogramm existiert und im RESET-Telegramm „Betrieb mit Anwesenheit“ gesetzt ist, wird jede Anwesenheitsmeldung wie ein unerwartetes Telegramm betrachtet (siehe Kapitel 3.3.12 Funktion moby_unexpect).

3.3.11 Funktion moby_diagnose

mobyErr_t moby_diagnose (mobyHandle_t handle, unsigned char mode, mobyDiagnose_t *diagnose, HANDLE sync, mobyErr_t *err);

Mit dieser Funktion lesen Sie Diagnosedaten aus dem SLG aus.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer oder der Ethernet-Schnittstelle.
mode	unsigned char	02 hex = letzten n Funktionsaufrufe anfordern 03 hex = letzten n Fehlermeldungen anfordern 04 hex = letzten n identifizierten MDS anfordern
diagnose	mobyDiagnose_t *	Pointer auf die Struktur, in die die gelesene Diagnoseinformation abzulegen ist.
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Es ist jeweils nur ein ausstehendes Diagnosetelegramm erlaubt. Jede Diagnoseanfrage, die zu einem Zeitpunkt abgesetzt wird, zu dem noch eine unbeantwortete Diagnoseanfrage besteht, wird mit einem entsprechenden Fehler zurückgewiesen. Die Diagnoseinformation ist MOBY U-spezifisch. Die Daten sind erst gültig, wenn durch das Synchronisations-Handle der Empfang des Antworttelegramms signalisiert wird. Der Inhalt und die Bedeutung sind den Anhängen B.2.2.2.3 bis B.2.2.2.5 zu entnehmen.

3.3.12 Funktion moby_unexpect

mobyErr_t moby_unexpect (mobyHandle_t handle,
moby_UnexpCallback_t cbroutine);

Mit dieser Funktion wird festgelegt, wie unerwartete Nachrichten behandelt werden. Hat der Parameter „cbroutine“ den Wert NULL, so werden nach dem Aufruf dieser Routine alle unerwarteten Nachrichten ignoriert. Ist der Wert nicht NULL, so muss er der Adresse einer gültigen Callback-Routine entsprechen. Diese wird beim Eintreffen unerwarteter Nachrichten angesprungen und ermöglicht dem Anwenderprogramm, auf dieses Ereignis zu reagieren. Beim Ansprechen der Callback-Routine werden die ersten 3 Byte des unerwarteten Telegramms und die tatsächliche Telegrammlänge übergeben. In der Headerdatei MOBY_API.H oder MOBY_API_T.H sind diese Übergabewerte mit „typdef void (CALLBACK *moby_UnexpCALLback_t) (unsigned char ch1, unsigned char ch2, unsigned char ch3, int laenge);“ definiert (siehe Kapitel 4.1 oder Kapitel 4.3).

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer oder der Ethernet-Schnittstelle.
cbroutine	moby_UnexpCallback_t	Callback-Routine für unerwartete Telegramme von SLG, ASM oder SIM.

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Unerwartete Telegramme, die eindeutig als Fehlermeldungen interpretiert werden können (wie z. B. die Meldung eines ungültigen Telegramms), werden als Fehler gewertet und behandelt wie in Kapitel 3.1.5 beschrieben.

3.4 MDS-Funktionen

MOBY API bietet MDS-Funktionen in zwei Varianten an:

- „ungezielte“ MDS-Funktionen und
- „gezielte“ MDS-Funktionen.

Mit den „ungezielten“ MDS-Funktionen:

- moby_read,
- moby_getID,
- moby_write,
- moby_init,
- moby_statusMDS,
- moby_readOTP und
- moby_writeOTP

lesen Sie Daten von einem beliebigen MDS oder schreiben Sie Daten auf einen beliebigen MDS. Sie habe keine Möglichkeit eine eindeutige Zuordnung zu einem MDS zu treffen. Deshalb darf sich nur ein MDS zum Zeitpunkt des Funktionsaufrufs und der Funktionsdurchführung im Antennenfeld befinden. Ein Multitag-Betrieb ist nicht möglich.

Diese Funktionen werden in Abhängigkeit von dem bei der Funktion moby_start übergebenen Parameter eccON, ohne oder mit ECC-Korrektur, durchgeführt (siehe Kapitel 3.3.1).

Welche Adressbereiche auf dem MDS von den einzelnen MOBY-Systemen gelesen und beschrieben werden können, ist aus den entsprechenden Dokumentationen /01/ bis /06/ oder auch Anhang A.8 für ASM 424/724/824 zu entnehmen.

Welche Funktionen mit welcher Parameterversorgung bei den MOBY-Systemen eingesetzt werden können, ist der Tabelle 3-1 und den Kapiteln 3.4.1 bis 3.4.6 zu entnehmen.

Achtung

Die „ungezielten“ MDS-Funktionen können nur über die serielle Schnittstelle betrieben werden (nicht über die Ethernet-Schnittstelle).

Mit den „gezielten“ MDS-Funktionen:

- moby_s_read,
- moby_s_getID,
- moby_s_write,
- moby_s_init,
- moby_s_copy,
- moby_s_statusMDS,

- moby_s_readOTP und
- moby_s_writeOTP

lesen Sie Daten von einem bestimmten MDS oder schreiben Sie Daten auf einen bestimmten MDS. Sie können anhand der MDS-Identifikationsnummer (Seriennummer) eine eindeutige Zuordnung zu einem MDS zu treffen. Wenn Sie bei diesen Funktionen, außer moby_s_getID und moby_s_copy, keine MDS-Nummer vorgeben, ist das Verhalten wie bei den „ungezielten“ Funktionsaufrufen.

Mit der Funktion moby_s_getID ermitteln Sie die Identnummer(n) von dem(den) im Antennenfeld befindlichen MDS. Sie können mehrere MDS im Antennenfeld bearbeiten, Multitag-Betrieb ist möglich. Zusätzlich bietet Ihnen die Funktion moby_s_copy die Möglichkeit, Daten von einem MDS (Quelle) auf einen anderen MDS (Ziel) zu kopieren, ohne über die Applikation zu gehen.

Die „gezielten“ MDS-Funktionen (das heißt Multitag-Betrieb ist möglich) können nur bei MOBY U eingesetzt werden. Bei der Funktion moby_start muss der Parameter type mit MOBY_Uc (MOBY U-Befehle für Singletag- und Multitag-Betrieb) gesetzt sein. Welche Funktionen mit welcher Parameterversorgung bei MOBY U eingesetzt werden können, ist der Tabelle 3-1 und den Kapiteln 3.4.8 bis 3.4.15 zu entnehmen.

Tabelle 3-4 Zuordnung der MDS-Funktionen zu den Schnittstellen

Funktion	Serielle Schnittstelle	Ethernet-Schnittstelle
moby_read	X	–
moby_getID	X	–
moby_write	X	–
moby_init	X	–
moby_statusMDS	X	–
moby_readOTP	X	–
moby_writeOTP	X	–
moby_s_read	X	X
moby_s_getID	X	X
moby_s_write	X	X
moby_s_init	X	X
moby_s_copy	X	X
moby_s_statusMDS	X	X
moby_s_readOTP	X	X
moby_s_writeOTP	X	X

3.4.1 Funktion moby_read

mobyErr_t moby_read (mobyHandle_t handle, uInt mdsAddress, unsigned char *data, uInt length, HANDLE sync, mobyErr_t *err);

Mit der Funktion moby_read werden Daten von dem MDS gelesen, der sich im Antennenfeld des SLG, der SLA oder des SIM befindet. Es ist ein „ungezielter“ Leseaufruf, da der MDS nicht anhand einer ID-Nummer identifizierbar ist.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer.
mdsAddress	uInt	Anfangsadresse der zu lesenden Daten auf dem MDS. ≥ 0 0 bis maximale Länge der Nutzdaten minus 1. Die Adresse plus Datenlänge muss kleiner sein als die Endadresse auf dem MDS.
data	unsigned char *	Pointer auf den Puffer, in dem die gelesenen Daten abzulegen sind.
length	uInt	Anzahl der zu lesenden Bytes. > 0 1 bis maximal 248 Byte
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Die eingelesenen Daten werden in den durch die Parameter „data“ und „length“ spezifizierten Puffer abgelegt. Diese Daten werden erst gültig, wenn durch das Synchronisations-Handle signalisiert wird, dass das Antworttelegramm mit den gelesenen Daten eingetroffen ist.
Bei den ASM 724/ASM 424 können mit einem Telegramm nur maximal 234 Byte und beim ASM 824 maximal 192 Byte gelesen werden.

3.4.2 Funktion moby_getID

mobyErr_t moby_getID (mobyHandle_t handle, unsigned char *idData, uInt idLength, HANDLE sync, mobyErr_t *err);

Mit der Funktion moby_getID wird die Identnummer (Seriennummer) des MDS gelesen. Das ist nur bei MDS von MOBY E, MOBY F und MOBY U möglich.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer.
idData	unsigned char *	Pointer auf den Puffer, in dem die gelesene ID-Nummer abzulegen ist.
idLength	uInt	Länge der ID-Nummer des MDS in Bytes. > 0 Länge der ID-Nummer : • MOBY E: ID-Länge = 4 Byte • MOBY F, MDS F1xx: ID-Länge = 5 Byte • MOBY F, MDS F4xx: ID-Länge = 4 Byte • MOBY U ID-Länge = 4 Byte
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Die Anfangsadresse der ID auf dem MDS wird intern versorgt.

- MOBY E: Anfangsadresse = 1FF0 hex
- MOBY F, MDS F1xx: Anfangsadresse = 0000 hex
- MOBY F, MDS F4xx: Anfangsadresse = 0000 hex

Die eingelesenen Daten werden in den durch die Parameter „idData“ und „idLength“ spezifizierten Puffer abgelegt. Diese Daten werden erst gültig, wenn durch das Synchronisations-Handle signalisiert wird, dass das Antworttelegramm mit den gelesenen Daten eingetroffen ist.

3.4.3 Funktion moby_write

mobyErr_t moby_write (mobyHandle_t handle, uInt mdsAddress, unsigned char *data, uInt length, HANDLE sync, mobyErr_t *err);

Mit der Funktion moby_write werden die Daten auf den MDS geschrieben, der sich im Antennenfeld des SLG, der SLA oder des SIM befindet. Es ist ein „ungezielter“ Schreibaufruf.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer.
mdsAddress	uInt	Anfangsadresse auf dem MDS für die zu schreibenden Daten. ≥ 0 0 bis maximale Länge der Nutzdaten minus 1. Die Adresse plus Datenlänge muss kleiner sein als die Endadresse auf dem MDS.
data	unsigned char *	Pointer auf den Puffer, in dem die zu schreibenden Daten abgelegt sind.
length	uInt	Anzahl der zu schreibenden Bytes. > 0 1 bis maximal 248 Byte
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Bei den ASM 724/ASM 424 können mit einem Telegramm nur maximal 234 Byte und beim ASM 824 maximal 192 Byte geschrieben werden.

3.4.4 Funktion moby_init

mobyErr_t moby_init (mobyHandle_t handle , unsigned char setVal, uInt mdsLength, HANDLE sync, mobyErr_t *err);

Mit der Funktion moby_init wird der MDS initialisiert (mit einem Datum vorbesetzt), der sich im Antennenfeld des SLG, der SLA oder des SIM befindet. Es ist ein „ungezielter“ Initialisierungsaufruf.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer.
setVal	unsigned char	1 Byte in Hex-Format, mit dem der gesamte MDS vorbesetzt werden soll. 00 hex bis FF hex
mdsLength	uInt	Anzahl der zu löschenden Bytes auf dem MDS. Die Anzahl der zu löschenden Bytes (Länge der Nutzdaten) ist der technischen Beschreibung zu entnehmen.
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Wenn die Anzahl der zu löschenden Bytes kleiner als die Länge der Nutzdaten des MDS ist, so wird der Speicher des MDS mit dem vorgegebenen Datum von Adresse 0 bis zur Adresse (Länge –1) beschrieben. Der restliche Speicher bleibt unverändert.

3.4.5 Funktion moby_statusMDS

mobyErr_t moby_statusMDS (mobyHandle_t handle, mobyStatusMDS_t *statusMDS, unsigned char mode, unsigned char cweek, unsigned char year, HANDLE sync, mobyErr_t *err);

Mit dieser Funktion werden Status- und Diagnosedaten eines MDS angefordert.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer.
statusMDS	mobyStatusMDS_t *	Pointer auf die Struktur, in der die Status- und Diagnoseinformation des Antworttelegramms abzulegen ist.
mode	unsigned char	00 hex = Status- und Diagnosedaten eines MDS anfordern
cweek	unsigned char	01 bis 35 hex = aktuelle Kalenderwoche (KW 1 bis KW 53) für die Bestimmung der restlichen Batterielebensdauer FF hex = ohne Bestimmung der restlichen Batterielebensdauer
year	unsigned char	01 bis 63 hex = die letzten zwei Stellen der aktuellen Jahreszahl (beginnend mit 01) für die Bestimmung der restlichen Batterielebensdauer FF hex = ohne Bestimmung der restlichen Batterielebensdauer
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Der Funktionsablauf ist von den Feldern cweak und year abhängig:
a) Ist der jeweilige Wert in beiden Feldern innerhalb des Wertebereiches, dann wird im Antworttelegramm die restliche Batterielebensdauer ausgegeben.
b) Haben beide den Wert FF hex, dann kann die restliche Batterielebensdauer nicht errechnet werden, sie wird mit FFFF hex Tagen angegeben. Die Statusdaten des MDS werden aber trotzdem ausgegeben.

Nach dem Aufruf stehen im Puffer die Status- und Diagnosedaten des MDS. Die Darstellungsform ist der MOBY U-Dokumentation /06/ zu entnehmen.

3.4.6 Funktion moby_readOTP

mobyErr_t moby_readOTP (mobyHandle_t handle, unsigned char *data, HANDLE sync, mobyErr_t *err);

Mit dieser Funktion kann der OTP-Speicher (One Time Programmable) mit 16 Byte (= 128 Bit) Länge gelesen werden.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer.
data	unsigned char *	Pointer auf den Puffer, in dem die aus dem OTP-Speicher gelesenen Daten abzulegen sind. Der Puffer muss eine Mindestlänge von 16 Byte haben.
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Der OTP-Speicher hat eine Länge von 16 Byte (= 128 Bit) und kann nur in einem Stück gelesen werden. Das heißt, die Funktion moby_readOTP liest grundsätzlich 16 Byte.
Der Puffer für die gelesenen Daten muss eine Mindestlänge von 16 Byte haben, sonst wird Ihnen Speicherbereich überschrieben.

3.4.7 Funktion moby_writeOTP

mobyErr_t moby_writeOTP (mobyHandle_t handle, unsigned char *data, HANDLE sync, mobyErr_t *err);

Mit dieser Funktion kann der OTP-Speicher (One Time Programmable) mit 16 Byte (= 128 Bit) Länge einmalig beschrieben werden.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer.
data	unsigned char *	Pointer auf den Puffer, in dem die in den OTP-Speicher zu schreibenden Daten abgelegt sind. Der Puffer muss eine Mindestlänge von 16 Byte haben.
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Der OTP-Speicher hat eine Länge von 16 Byte (= 128 Bit) und kann nur einmalig in einem Stück beschrieben werden. Das heißt, die Funktion moby_writeOTP schreibt grundsätzlich 16 Byte.

3.4.8 Funktion moby_s_read

mobyErr_t moby_s_read (mobyHandle_t handle, uInt *idData, uInt mdsAddress, unsigned char *data, uInt length, HANDLE sync, mobyErr_t *err);

Mit dieser Funktion lesen Sie „gezielt“ oder „ungezielt“ Daten von einem MDS, der sich im Antennenfeld des SLG U92 befindet.

- Es ist ein „gezielter“ Leseaufruf, wenn der Aufruf mit Identnummer des MDS abgesetzt wird. Mehrere MDS dürfen sich im Antennenfeld befinden. Die Identnummer des MDS können Sie über die Funktion GET ermitteln.
- Es ist ein „ungezielter“ Leseaufruf, wenn der Aufruf ohne Identnummer des MDS abgesetzt wird. Es darf sich nur ein MDS im Antennenfeld befinden.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer oder der Ethernet-Schnittstelle.
idData	uInt *	Wert von 2^0 bis 2^{32}-1 Identnummer des MDS 0 Wenn sich nur ein MDS im Feld befindet und ein „ungezielter“ Leseaufruf erfolgen soll.
mdsAddress	uInt	Anfangsadresse der zu lesenden Daten auf dem MDS. ≥ 0 0 bis maximale Länge der Nutzdaten minus 1. Die Adresse plus Datenlänge muss kleiner sein als die Endadresse auf dem MDS.
data	unsigned char *	Pointer auf den Puffer, in dem die gelesenen Daten abzulegen sind.
length	uInt	Anzahl der zu lesenden Bytes. > 0 1 bis maximal 244 Byte
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Der Befehl moby_s_read darf nur an das SLG U92 abgesetzt werden, wenn noch kein Befehl beim SLG U92 ansteht. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.
Wenn sich beim „gezielten“ Aufruf der MDS mit der vorgegebenen Identnummer nicht in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.
Wenn sich beim „ungezielten“ Aufruf kein MDS in der Zone 1 befindet, so bleibt der Befehl anstehen, bis ein MDS in die Zone 1 oder der Befehl RESET kommt.
Wenn sich beim „ungezielten“ Aufruf mehr als ein MDS in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.

Achtung

Die eingelesenen Daten werden in den durch die Parameter „data“ und „length“ spezifizierten Puffer abgelegt. Diese Daten werden erst gültig, wenn durch das Synchronisations-Handle signalisiert wird, dass das Antworttelegramm mit den gelesenen Daten eingetroffen ist.

Achtung

Beim SLG U92 mit ASM 480 (TCP/IP) können mit einem Telegramm nur maximal 228 Byte gelesen werden.

3.4.9 Funktion moby_s_getID

mobyErr_t moby_sgetID (mobyHandle_t handle, mobyMtget_t *getInfo, int address, uInt length, HANDLE sync, mobyErr_t *err);

Mit dieser Funktion fragen Sie ab, welche MDS sich im Feld befinden. Sie können gleichzeitig mit der Abfrage Daten von diesen MDS lesen.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer oder der Ethernet-Schnittstelle.
getInfo	mobyMtget_t *	Pointer auf die Struktur, in die die Anzahl der abgefragten MDS, die Identnummern der MDS und wenn gefordert, die gelesenen Daten der MDS abzulegen sind.
address	int	Anfangsadresse der zu lesenden Daten auf dem MDS. ≥ 0 0 bis maximale Länge der Nutzdaten minus 1. Die Adresse plus Datenlänge muss kleiner sein als die Endadresse auf dem MDS. - 16 = FFF0 hex Anfangsadresse des OTP-Speichers
length	uInt	Anzahl der zu lesenden Bytes. ≥ 0 0 bis maximal Länge x x = (242 – (4 * Pulkgröße)) / Pulkgröße 16 Länge des OTP-Speichers
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Der Befehl moby_s_getID darf nur an das SLG U92 abgesetzt werden, wenn noch kein Befehl beim SLG U92 ansteht. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.

Achtung

Die 128 Bit-Anwenderinformation im OTP-Speicher wird mit der Startadresse –16 (FFF0 hex) adressiert. Die 128 Bit-Anwenderinformation wird mit dem Befehl WRITE in den MDS geschrieben. Beim GET-Aufruf muss die vollständige Information mit 128 Bit Länge angefordert werden.

Achtung

Beim SLG U92 mit ASM 480 (TCP/IP) können mit einem Telegramm nur maximal 226 Byte (MDS-Identnummern und Nutzdaten) gelesen werden. Das bedeutet: Parameter length = x = (226 – (4 * Pulkgröße)) / Pulkgröße.

3.4.10 Funktion moby_s_write

mobyErr_t moby_swrite (mobyHandle_t handle, uInt *idData, uInt mdsAddress, unsigned char *data, uInt length, HANDLE sync, mobyErr_t *err);

Mit dieser Funktion schreiben Sie „gezielt“ oder „ungezielt“ Daten auf einen MDS, der sich im Antennenfeld des SLG U92 befindet.

- Es ist ein „gezielter“ Schreibaufruf, wenn der Aufruf mit Identnummer des MDS abgesetzt wird. Mehrere MDS dürfen sich im Antennenfeld befinden. Die Identnummer des MDS können Sie über die Funktion GET ermitteln.
- Es ist ein „ungezielter“ Schreibaufruf, wenn der Aufruf ohne Identnummer des MDS abgesetzt wird. Es darf sich nur ein MDS im Antennenfeld befinden.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer oder der Ethernet-Schnittstelle.
idData	uInt *	Wert von 2^0 bis 2^{32}-1 Identnummer des MDS 0 Wenn sich nur ein MDS im Feld befindet und ein „ungezielter“ Schreibaufruf erfolgen soll.
mdsAddress	uInt	Anfangsadresse auf dem MDS für die zu schreibenden Daten. ≥ 0 0 bis maximale Länge der Nutzdaten minus 1. Die Adresse plus Datenlänge muss kleiner sein als die Endadresse auf dem MDS.
data	unsigned char *	Pointer auf den Puffer, in dem die zu schreibenden Daten abgelegt sind.
length	uInt	Anzahl der zu schreibenden Bytes. ≥ 0 1 bis maximal 244 Byte
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Der Befehl moby_s_write darf nur an das SLG U92 abgesetzt werden, wenn noch kein Befehl beim SLG U92 ansteht. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.
Wenn sich beim „gezielten“ Aufruf der MDS mit der vorgegebenen Identnummer nicht in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.
Wenn sich beim „ungezielten“ Aufruf kein MDS in der Zone 1 befindet, so bleibt der Befehl anstehen, bis ein MDS in die Zone 1 oder der Befehl RESET kommt.
Wenn sich beim „ungezielten“ Aufruf mehr als ein MDS in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.

Achtung

Beim SLG U92 mit ASM 480 (TCP/IP) können mit einem Telegramm nur maximal 228 Byte geschrieben werden.

3.4.11 Funktion moby_s_init

mobyErr_t moby_sinit (mobyHandle_t handle, uInt *idData, unsigned char setVal, uInt mdsLength, HANDLE sync, mobyErr_t *err);

Mit dieser Funktion initialisieren Sie „gezielt“ oder „ungezielt“ einen MDS mit einem Bitmuster. Der MDS befindet sich im Antennenfeld des SLG U92.

- Es ist ein „gezielter“ Initialisierungsaufruf, wenn der Aufruf mit Identnummer des MDS abgesetzt wird. Mehrere MDS dürfen sich im Antennenfeld befinden. Die Identnummer des MDS können Sie über die Funktion GET ermitteln.
- Es ist ein „ungezielter“ Initialisierungsaufruf, wenn der Aufruf ohne Identnummer des MDS abgesetzt wird. Es darf sich nur ein MDS im Antennenfeld befinden.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer oder der Ethernet-Schnittstelle.
idData	uInt *	Wert von 2^0 bis 2^{32}-1 Identnummer des MDS 0 Wenn sich nur ein MDS im Feld befindet und ein „ungezielter“ Initialisierungsaufruf erfolgen soll.
setVal	unsigned char	1 Byte in Hex-Format, mit dem der gesamte MDS vorbesetzt werden soll. 00 hex bis FF hex
mdsLength	uInt	Anzahl der zu löschenden Bytes auf dem MDS. Die Anzahl der zu löschenden Bytes (Länge der Nutzdaten) ist der technischen Beschreibung zu entnehmen.
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Der Befehl moby_s_init darf nur an das SLG U92 abgesetzt werden, wenn noch kein Befehl beim SLG U92 ansteht. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.
Wenn sich beim „gezielten“ Aufruf der MDS mit der vorgegebenen Identnummer nicht in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.
Wenn sich beim „ungezielten“ Aufruf kein MDS in der Zone 1 befindet, so bleibt der Befehl anstehen, bis ein MDS in die Zone 1 oder der Befehl RESET kommt.
Wenn sich beim „ungezielten“ Aufruf mehr als ein MDS in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.

3.4.12 Funktion moby_s_copy

mobyErr_t moby_s_copy (mobyHandle_t handle, uInt idData1, uInt addr1, uInt idData2, uInt addr2, uInt len, HANDLE sync, mobyErr_t *err);

Mit dieser Funktion kopieren Sie „gezielt“ Daten: einen Datenbereich oder den kompletten Datenträgerinhalt von einem auf einen anderen MDS. Mit dieser Funktion können Daten ohne Umweg über das Anwenderprogramm von einem MDS (Quelle) auf einen anderen MDS (Ziel) geschrieben, d. h., kopiert werden.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer oder der Ethernet-Schnittstelle.
idData1	uInt	Wert von 2^0 bis 2^{32}-1 Identnummer des MDS 1 (Quelle), von dem kopiert werden soll.
addr1	int	0 bis maximale Länge der Nutzdaten minus 1. Anfangsadresse der zu kopierenden Daten von dem MDS 1. Die Adresse plus Datenlänge muss kleiner sein als die Endadresse.
idData2	uInt	Wert von 2^0 bis 2^{32}-1 Identnummer des MDS 2 (Ziel), auf den kopiert werden soll.
addr2	int	0 bis maximale Länge der Nutzdaten minus 1. Anfangsadresse auf dem MDS 2, ab der die Daten geschrieben werden sollen. Die Adresse plus Datenlänge muss kleiner sein als die Endadresse.
len	uInt	Anzahl der zu kopierenden Bytes. 1 bis maximale Länge der Nutzdaten
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Der Befehl moby_s_copy darf nur an das SLG U92 abgesetzt werden, wenn noch kein Befehl beim SLG U92 ansteht. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.
Wenn sich die zwei angegebenen MDS nicht in der Zone 1 befinden, so wird der Befehl mit einem Fehler abgebrochen.

Achtung

Der OTP-Speicher kann mit dem Befehl moby_s_copy nicht kopiert werden.

3.4.13 Funktion moby_s_statusMDS

mobyErr_t moby_s_statusMDS (mobyHandle_t handle, uInt *idData, mobyStatusMDS_t *statusMDS, unsigned char mode, unsigned char cweek, unsigned char year, HANDLE sync, mobyErr_t *err);

Mit dieser Funktion fragen Sie „gezielt“ oder „ungezielt“ Status- und Diagnosedaten von einem MDS ab, der sich im Antennenfeld des SLG U92 befindet.

- Es ist ein „gezielter“ Leseaufruf, wenn der Aufruf mit Identnummer des MDS abgesetzt wird. Mehrere MDS dürfen sich im Antennenfeld befinden. Die Identnummer des MDS können Sie über die Funktion GET ermitteln.
- Es ist ein „ungezielter“ Leseaufruf, wenn der Aufruf ohne Identnummer des MDS abgesetzt wird. Es darf sich nur ein MDS im Antennenfeld befinden.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer oder der Ethernet-Schnittstelle.
idData	uInt *	Wert von 2^0 bis 2^{32}-1 Identnummer des MDS 0 Wenn sich nur ein MDS im Feld befindet und eine „ungezielte“ Statusabfrage erfolgen soll.
statusMDS	mobyStatusMDS_t *	Pointer auf die Struktur, in der die Status- und Diagnoseinformation des Antworttelegramms abzulegen ist.
mode	unsigned char	00 hex = Status- und Diagnosedaten eines MDS anfordern
cweek	unsigned char	01 bis 35 hex = aktuelle Kalenderwoche (KW 1 bis KW 53) für die Bestimmung der restlichen Batterielebensdauer FF hex = ohne Bestimmung der restlichen Batterielebensdauer
year	unsigned char	01 bis 63 hex = die letzten zwei Stellen der aktuellen Jahreszahl (beginnend mit 01) für die Bestimmung der restlichen Batterielebensdauer FF hex = ohne Bestimmung der restlichen Batterielebensdauer
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.

Parameter	Typ	Beschreibung
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.

Der Funktionsablauf ist von den Feldern cweek und year abhängig:

a) Ist der Wert in beiden Feldern innerhalb des Wertebereiches, dann wird in der Antwort die restliche Batterielebensdauer ausgegeben.

b) Ist einer der Werte außerhalb des angegebenen Wertebereiches, dann kann die restliche Batterielebensdauer nicht errechnet werden und die Funktion wird mit einem Fehler abgebrochen.
Sind beide Werte mit FF hex Tagen vorgegeben, so wird die restliche Batterielebensdauer nicht berechnet und in der Quittung wird als Batterielebensdauer FFFF hex angegeben.

Wenn sich beim „gezielten“ Aufruf der MDS mit der vorgegebenen Identnummer nicht in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.
Wenn sich beim „ungezielten“ Aufruf kein MDS in der Zone 1 befindet, so erfolgt eine Fehlermeldung.
Wenn sich beim „ungezielten“ Aufruf mehr als ein MDS in der Zone 1 befindet, so wird der Befehl mit Fehler abgebrochen.

3.4.14 Funktion moby_s_readOTP

mobyErr_t moby_s_readOTP (mobyHandle_t handle, uInt *idData, unsigned char *data, HANDLE sync, mobyErr_t *err);

Mit dieser Funktion lesen Sie einmalig „gezielt“ oder „ungezielt“ den OTP-Speicher (One Time Programmable) mit 16 Byte (= 128 Bit) Länge.

- Es ist ein „gezielter“ Leseaufruf, wenn der Aufruf mit Identnummer des MDS abgesetzt wird. Mehrere MDS dürfen sich im Antennenfeld befinden. Die Identnummer des MDS können Sie über die Funktion GET ermitteln.
- Es ist ein „ungezielter“ Leseaufruf, wenn der Aufruf ohne Identnummer des MDS abgesetzt wird. Es darf sich nur ein MDS im Antennenfeld befinden.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer oder der Ethernet-Schnittstelle.
idData	uInt *	Wert von 2^0 bis 2^{32}-1 Identnummer des MDS 0 Wenn sich nur ein MDS im Feld befindet und eine „ungezielter“ Leseaufruf erfolgen soll.
data	unsigned char *	Pointer auf den Puffer, in dem die aus dem OTP-Speicher gelesenen Daten abzulegen sind. Der Puffer muss eine Mindestlänge von 16 Byte haben.
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Der Befehl moby_s_readOTP darf nur an das SLG U92 abgesetzt werden, wenn noch kein Befehl beim SLG U92 ansteht. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.
Wenn sich beim „gezielten“ Aufruf der MDS mit der vorgegebenen Identnummer nicht in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.
Wenn sich beim „ungezielten“ Aufruf kein MDS in der Zone 1 befindet, so bleibt der Befehl anstehen, bis ein MDS in die Zone 1 oder der Befehl RESET kommt.
Wenn sich beim „ungezielten“ Aufruf mehr als ein MDS in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.

Achtung

Die 128 Bit-Anwenderinformation wird mit dem Befehl WRITE in den MDS geschrieben. Beim Leseaufruf muss die vollständige Information mit 128 Bit Länge angefordert werden.

3.4.15 Funktion moby_s_writeOTP

mobyErr_t moby_s_writeOTP (mobyHandle_t handle, uInt *idData, unsigned char *data, HANDLE sync, mobyErr_t *err);

Mit dieser Funktion beschreiben Sie einmalig „gezielt“ oder „ungezielt“ den OTP-Speicher (One Time Programmable) mit 16 Byte (= 128 Bit) Länge.

- Es ist ein „gezielter“ Schreibaufruf, wenn der Aufruf mit Identnummer des MDS abgesetzt wird. Mehrere MDS dürfen sich im Antennenfeld befinden. Die Identnummer des MDS können Sie über die Funktion GET ermitteln.
- Es ist ein „ungezielter“ Schreibaufruf, wenn der Aufruf ohne Identnummer des MDS abgesetzt wird. Es darf sich nur ein MDS im Antennenfeld befinden.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer oder der Ethernet-Schnittstelle.
idData	uInt *	Wert von 2^0 bis 2^{32}-1 Identnummer des MDS 0 Wenn sich nur ein MDS im Feld befindet und eine „ungezielter“ Schreibaufruf erfolgen soll.
data	unsigned char *	Pointer auf den Puffer, in dem die in den OTP-Speicher zu schreibenden Daten abgelegt sind. Der Puffer muss eine Mindestlänge von 16 Byte haben.
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Der Befehl moby_s_writeOTP darf nur an das SLG U92 abgesetzt werden, wenn noch kein Befehl beim SLG U92 ansteht. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.
Wenn sich beim „gezielten“ Aufruf der MDS mit der vorgegebenen Identnummer nicht in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.
Wenn sich beim „ungezielten“ Aufruf kein MDS in der Zone 1 befindet, so bleibt der Befehl anstehen, bis ein MDS in die Zone 1 oder der Befehl RESET kommt.
Wenn sich beim „ungezielten“ Aufruf mehr als ein MDS in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.

Achtung

Der OTP-Speicher kann nur einmal beschrieben werden. Bei diesem Schreibaufruf muss die vollständige Information mit 128 Bit Länge übergeben werden. Ein weiterer Schreibversuch wird mit Fehlermeldung abgewiesen.

3.5 DE-/DA-Funktionen

Mit den DE-/DA-Funktionen moby_readDE und moby_writeDA können Sie digitale Eingänge (DE) abfragen (lesen) und digitale Ausgänge (DA) setzen (schreiben). Sie sind nur bei SLG an ASM 420 und SIM 41 möglich.

3.5.1 Funktion moby_readDE

mobyErr_t moby_readDE (mobyHandle_t handle, mobyDEDA_t *deda, HANDLE sync, mobyErr_t *err);

Diese Funktion fragt digitale Eingänge (DE) ab.

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer.
deda	mobyDEDA_t *	Pointer auf die Struktur, in die die gelesenen Signalzustände abzulegen sind.
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Nach dem Aufruf stehen im Puffer die Signalzustände der digitalen Eingänge. Die Anzahl und die Darstellungsform sind MOBY-systemspezifisch und der entsprechenden MOBY-Dokumentation /01/ oder /02/ zu entnehmen.

3.5.2 Funktion moby_writeDA

mobyErr_t moby_writeDA (mobyHandle_t handle, mobyDEDA_t *deda, HANDLE sync, mobyErr_t *err);

Diese Funktion setzt digitale Ausgänge (DA).

Parameter	Typ	Beschreibung
handle	mobyHandle_t	Handle der seriellen Schnittstelle und der Kanalnummer.
deda	mobyDEDA_t *	Pointer auf den Puffer, in dem die zu schreibenden Signalzustände abgelegt sind.
sync	HANDLE	Handle zur Synchronisation mit dem Antworttelegramm. Initialisiertes Windows-Event, anhand dessen die Applikation auf das Antworttelegramm wartet.
err	mobyErr_t *	Pointer auf die Struktur, in der das Statusbyte des Antworttelegramms mit eventuellem Fehlercode und Schnittstellenfehler abzulegen ist (siehe Kapitel 3.7.2).

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Achtung

Im Puffer sind die Signalzustände der zu setzenden digitalen Ausgänge und die zugehörigen Bedingungen. Die Anzahl und die Darstellungsform sind MOBY-systemspezifisch und der entsprechenden MOBY-Dokumentation /01/ oder /02/ zu entnehmen.

3.6 Funktion moby_version

mobyErr_t moby_version (int *major, int *minor);

Mit der Funktion moby_version fragen Sie die Version der dynamischen Link-Bibliothek MOBY_API.DLL oder MOBY_API_T.DLL ab. Die Versionsnummer wird in zwei Teilen zurückgeliefert, der Hauptnummer xx und der Unternummer yy. Sie ist zu „xx.yy“ zusammenzustellen.

Parameter	Typ	Beschreibung
major	int *	Speicherwort für den Wert xx der Versionnummer
minor	int *	Speicherwort für den Wert yy der Versionnummer

Rückgabewert:

≥ 0 kein Fehler, Befehl ausgeführt
< 0 Schnittstellenfehler (siehe Kapitel 3.7.1).

Beispiel: Die Version 1.0 erscheint als xx = 1 und yy = 0.

3.7 Funktionsfehler

3.7.1 Schnittstellenfehler als Rückgabewert

Die Schnittstellenfehler, die von den beschriebenen Funktionen als Rückgabewert geliefert werden, sind:

- Fehler direkt aus der DLL MOBY_API oder MOBY_API_T,
- Fehler aus dem Treiber 3964R (nur serielle Ankopplung an PC) oder
- Fehler aus dem SLG, ASM mit SLG oder SLA oder SIM.

Die Fehler aus dem Treiber 3964R sind in zwei Klassen unterteilt:

- Fehler-Statusmeldungen: Sie signalisieren Statusfehler, die während der Übertragung von Telegrammen auftreten. Sie hängen größtenteils mit der Schnittstelle bzw. lokalen Systemressourcen zusammen.
- System-Fehlermeldungen: Sie zeigen das Fehlschlagen von Funktionsaufrufen an und bedeuten meist einen gravierenden Fehler im System und/oder in den Konfigurationseinstellungen.

Der Rückgabewert ist vom Typ Long Integer und grundsätzlich negativ und nach folgendem Schema aufgebaut:

Rückgabewert = Fehlercode **OR** Fehlerblock **OR** Fehlernummer

Die Werte von Fehlercode, Fehlerblock und Fehlernummer sind ODER-verknüpft und können folgende Werte annehmen:

Fehlercode:	0x80 00 00 00	
Fehlerblock:	0x1 00 00	Fehler direkt aus der DLL MOBY_API
	0x2 00 00	Fehler aus dem Treiber 3964R
	0x3 00 00	Fehler aus dem SLG, ASM mit SLG oder SLA oder SIM

Fehlernummern:

- MOBY_API oder MOBY_API_T

0x00 00	Fataler Fehler
0x00 01	Falsches Device-Handle
0x00 02	Falscher Parameter
0x00 03	Falsches/nicht passendes Antworttelegramm
0x00 04	Antworttelegramm hat falsche Länge
0x00 05	Matchpuffer voll
0x00 06	Falscher MDS-Typ
0x00 07	Keine weiteren Handle verfügbar
0x00 08	Windows-Systemfehler
0x00 09	Befehl durch „moby_stop“ abgebrochen
0x00 0A	Keine ID-Nummern bei MOBY I
0x00 0B	Handle noch offen
0x00 0C	Befehl durch Fehler abgebrochen (implizites „moby_stop“)
0x00 0D	Kanal bereits offen
0x00 0E	Kanal bereits gestartet
0x00 0F	Kanal noch nicht gestartet
0x00 10	Vorhergehende Statusabfrage noch nicht abgeschlossen
0x00 11	Diese Nummer (MOBY_ERR_NOTSUPPORTED) steht für Funktionen, die bei der gewählten Anschaltung nicht vorgesehen sind; z. B. moby_s_getID, wenn nicht MOBY U-SLG
0x00 12	Falsche IP-Adresse vergeben
0x00 13	Fehler beim Socket-Handle
0x00 14	Fehler beim Verbindungsaufbau TCP/IP
0x00 15	Ungültige Gerätekennung

Wenn eine der Fehlernummern 0x00 00; 0x00 01; 0x00 02; 0x00 05; 0x00 07; 0x00 12; 0x00 13; 0x00 14 oder 0x00 15 gemeldet wird, muss die entsprechende Schnittstelle, falls geöffnet, mit moby_close geschlossen und mit moby_open erneut geöffnet werden.

- Treiber 3964R

Fehler-Statusmeldungen

0x80 01	Zu viele Aufträge, Sendewarteschlange voll
0x80 02	Verbindungsaufbau erfolglos (Senden)
0x80 04	Erfolgloser Telegrammtransfer (Senden)
0x80 08	Empfangspuffer beim Empfang zu klein
0x80 10	Blockcheck-Fehler beim Empfangen
0x80 20	Timeout beim Verbindungsaufbau (Senden)
0x80 40	Timeout während eines Telegramms (Empfangen)
0x80 80	Timeout beim Verbindungsabbau (Senden)
0x81 00	COM-Meldung: Break
0x82 00	COM-Meldung: Paritätsfehler
0x84 00	COM-Meldung: Framingfehler
0x88 00	COM-Meldung: Overrun
0x90 00	Duplex-Konflikt (Senden)
0xC0 00	Unbekannter Systemfehler

System-Fehlermeldungen

0xA0 00	Kein Fehler
0xA0 02	Übergebenes Handle ungültig
0xA0 03	Nicht genug Speicher verfügbar
0xA0 04	Kein Platz im Sendepuffer
0xA0 05	Pufferüberlauf
0xA0 06	Fehler bei einer Operation auf einem globalen Event (OS-Fehler)
0xA0 07	Schnittstelle bereits geöffnet
0xA0 08	Schnittstelle nicht geöffnet
0xA0 09	Zur Zeit nicht verwendet
0xA0 0A	Zur Zeit nicht verwendet
0xA0 0B	Zur Zeit nicht verwendet
0xA0 0C	Zur Zeit nicht verwendet
0xA0 0D	Zur Zeit nicht verwendet
0xA0 0E	Zur Zeit nicht verwendet
0xA0 0F	Zur Zeit nicht verwendet
0xA0 10	Zur Zeit nicht verwendet
0xA0 11	Nachrichtenparameter ungültig
0xA0 12	Zur Zeit nicht verwendet
0xA0 13	Empfangsauftrag fehlerhaft
0xA0 14	Sendeauftrag fehlerhaft
0xA0 15	Zur Zeit nicht verwendet
0xA0 16	Zur Zeit nicht verwendet
0xA0 17	NAK während Senden eines Telegramms
0xA0 18	Zeichenempfang während Senden eines Telegramms (nicht NAK)
0xA0 64	Schnittstellen-Initialisierungsfehler
0xA0 65	Probleme beim Thread oder beim Öffnen/Schließen der COM-Schnittstelle
0xA0 66	Zur Zeit nicht verwendet
0xA0 67	Zur Zeit nicht verwendet
0xA0 68	Zur Zeit nicht verwendet
0xA0 69	Zur Zeit nicht verwendet
0xE0 01	Angesprochener Port ist nicht konfiguriert
0xE0 02	Fehler beim Zugriff auf die Windows-Registry
0xE0 03	Kein weiterer Port mehr zu öffnen

- SLG, ASM oder SIM Die möglichen Fehlernummern sind im Kap. 3.7.2 unter MOBY-Fehlercode aufgeführt.

Achtung

Wenn Windows die Kommunikation zwischen dem PC und dem ASM, SLG oder SIM unterbricht, in dem eine Überwachungszeit überschritten wird (siehe Kapitel 2.2.3), so kann es zu dem Fehler 0x000C aus dem C-Interface MOBY API oder zu dem Fehler 0xA017 direkt aus dem Treiber 3964R kommen. Zusätzlich kann es zu einem unerwarteten Telegramm führen, das nur bei Nutzung der Callback-Routine (siehe Kapitel 3.1.3 und 3.3.12) gemeldet wird. Dieses unerwartete Telegramm ist zu verwerfen.

- Nach dem Fehler 0x000C muss die Funktion moby_start abgesetzt und anschließend die fehlgeschlagene Funktion wiederholt werden. Die Funktion moby_stop ist nicht erforderlich. Falls Sie sie doch aufrufen, so kommt der Fehler 0x000F.
- Nach dem Fehler 0xA017 kann der Befehl, bei dem der Fehler auftrat, ohne moby_stop und moby_start direkt wiederholt werden.

Sollte bei moby_start ein Fehler auftreten, so ist die COM-Schnittstelle mit moby_close zu schließen und danach neu zu öffnen.

3.7.2 MOBY-Status

Nach der Ausführung der MOBY-Funktion (Antworttelegramm ist eingetroffen) ist das MOBY-Statusbyte des Antworttelegramms unter der mit dem Pointer „status“ adressierten Union „mobyErr_t“ abgelegt und kann als

- Struktur „mobyStatus_t“ oder
- Long Integer-Variable „error“

angesprochen werden (siehe Headerdatei MOBY_API.H, Kapitel 4.1 oder MOBY_API_T.H, Kapitel 4.3).
So kann die Fehler- und Statusinformation sowohl anhand von Strukturparametern einzeln (bitweise) als auch als eine Einheit (siehe auch Fehlernummern aus dem ASM oder SIM mit SLG oder SLA in Kapitel 3.7.1) ausgelesen werden.

Struktur „mobyStatus_t“

Tabelle 3-5 Strukturparameter des MOBY-Status

Strukturparameter (Variable)	Bedeutung
error	Fehlerkennung
dummy	Reserve
busyASM	MDS-Befehl an ASM aktiv/kein MDS-Befehl an ASM aktiv
anwMDS	Anwesenheitsmeldung
batMDS	Status Batteriespannung MDS (Batt. 1)
diagBatMDS507	Status Batteriespannung MDS 507/407 E (Batt. 2)
eccDone	ECC-Korrekturstatus auf MDS
errorCode	MOBY-Fehlercode

error

Fehlerkennung, die aussagt, ob ein Fehlercode vorliegt.
1: Fehler. „errorCode“ > 0.
0: Kein Fehler.

dummy

Reservebereich (-bits).

busyASM

MDS-Befehl an ASM aktiv/kein MDS-Befehl an ASM aktiv (entspricht dem Bit 0 im Byte ANW/Busy)
1: MDS-Befehl an ASM aktiv
0: kein MDS-Befehl an ASM aktiv

anwMDS

Die Anwesenheitsmeldung entspricht dem Bit 1 im Byte ANW/Busy und sagt aus, ob sich ein MDS im Feld des SLG/SLA befindet.
0: kein MDS im Feld
1: MDS im Feld

batMDS

Status Batteriespannung entspricht dem Bit 7 vom Statusbyte (Batt. 1) und gibt den Zustand der Dialogbatterie am MDS an.

1: Batterie auf MDS ist unter Schwellenwert abgefallen.
Bei MDS-Typen mit EEPROM-Speicher ist dieses Bit immer gesetzt.

diagBatMDS507

Status Dialogbatterie am MDS entspricht dem Bit 6 vom Statusbyte (Batt. 2) (nur für MDS 507/407 E).

1: Batterie unter Schwellenwert
Bei anderen MDS kann das Bit den Wert „0“ oder „1“ haben.

eccDone

Der ECC-Korrekturstatus entspricht dem Bit 5 vom Statusbyte und ist nur relevant, wenn in der Systemfunktion moby_start „mit ECC-Korrektur“ angegeben wurde.

1: ECC-Korrektur wurde durchgeführt
(die Daten im Ergebnistelegramm sind in Ordnung)

errorCode

Der MOBY-Fehlercode entspricht den Bits 0 bis 4 im MOBY-Statusbyte.

Tabelle 3-6 MOBY-Fehlercode

Fehler (hex)	Bedeutung	Zuordnung zum ASM/SIM/SLG					
		ASM 824	ASM 724	ASM 424	ASM 420	SIM 41	SLG U92
00	Kein Fehler. Antworttelegramm fehlerfrei.	x	x	x	x	x	x
01	ANW-Fehler: MDS aus Feld, wenn Befehl aktiv	x	x	x	x	x	x
02	ANW-Fehler: MDS ohne Befehl an SLG vorbei	x	x	x	x	x	–
	Ein anstehender MDS-Befehl wurde durch den Befehl „Antenne ausschalten“ abgebrochen.	–	–	–	–	–	x
03	Fehler in der Verbindung zum SLG/SIM/SLA	x	x	x	x	x	–
04	Fehler im Speicher des MDS (nicht initialisiert)	x	x	x	x	x	x
05	Befehl vom SLG/ASM nicht interpretierbar	x	x	x	x	x	x
06	Feldstörung am SLG/SIM/SLA	x	x	x	x	x	x
07	Zu viele Sendefehler	x	x	x	x	x	–
08	MDS meldet sehr oft CRC-Fehler	x	x	x	–	x	–
09	INIT: CRC-Fehler	x	x	x	x	x	–
0A	INIT: MDS lässt sich nicht initialisieren	x	x	x	x	x	–

Tabelle 3-6 MOBY-Fehlercode

Fehler (hex)	Bedeutung	Zuordnung zum ASM/SIM/SLG					
		ASM 824	ASM 724	ASM 424	ASM 420	SIM 41	SLG U92
0B	INIT: Timeout beim Initialisieren	x	x	x	x	x	–
	Speicher des MDS nicht korrekt lesbar	–	–	–	–	–	x
0C	INIT: Schreibfehler beim Initialisieren	x	x	x	x	x	x
	Wiederholtes Schreiben auf OTP-Speicher	–	–	–	–	–	x
0D	Adressfehler	x	x	x	x	x	x
0E	ECC-Betrieb: Daten im MDS sind falsch	x	x	x	x	x	–
0F	RESET-Meldung nach Spannungswiederkehr	x	x	x	x	x	–
10	NEXT-Befehl nicht zugelassen	x	x	x	x	x	x
11	Kurzschluss oder Überlastung der digitalen Ausgänge	x	x	x	–	–	–
12	Interner Firmwarefehler	x	x	x	–	–	–
13	Watchdog	x	x	x	–	–	–
	Es sind nicht genügend Puffer im SLG für die Speicherung des Befehls vorhanden	–	–	–	–	–	x
14	Firmwarefehler	x	x	x	–	–	x
	Watchdog-Meldung aus SLG U	–	–	–	–	–	x
15	Parametrierfehler	x	x	x	–	–	x
16	Ungeeignete Verbindungskonfiguration	x	x	x	–	–	–
17	Protokollfehler	x	x	x	–	–	–
18	Nur RESET-Befehl zulässig	x	x	x	–	–	x
19	vorheriger Befehl ist aktiv	–	–	–	x	x	x
	Pufferüberlauf, alle Telegrammpuffer belegt	x	x	x	–	–	–
1A	3964R-Fehler (Verbindung unterbrochen)	x	x	x	–	–	–
1B	Sendeauftrag im SLG zu häufig wiederholt; Datenverlust möglich; RESET-Befehl erforderlich	–	–	–	–	–	x
1C	Antenne ein-/ausgeschaltet	–	–	–	–	–	x

Tabelle 3-6 MOBY-Fehlercode

Fehler (hex)	Bedeutung	Zuordnung zum ASM/SIM/SLG					
		ASM 824	ASM 724	ASM 424	ASM 420	SIM 41	SLG U92
1D	zu wenig RAM auf dem ASM vorhanden	–	–	–	x	–	–
	Anzahl der MDS > Pulk	–	–	–	–	–	x
1E	Falsche Anzahl Zeichen im Telegramm	x	x	x	x	x	x
1F	SLG/ASM-Befehl mit RESET gelöscht	x	x	x	x	x	x

Long Integer-Variable „error“

Wenn die Fehler- und Statusinformation als Long Integer-Wert ausgelesen wird, müssen die einzelnen Informationen per Maskierung ausgewertet werden.

4 Headerdateien

In jede Applikation ist die Headerdatei

- MOBY_API.H für serielle Ankopplung an PC oder
- MOBY_API_T.H für Ankopplung an Ethernet

mit dem Präprozessorbefehl „#include“ einzubinden. Dadurch werden alle Funktionsaufrufe und Konstanten deklariert.

4.1 Headerdatei MOBY_API.H

```
// Headerfile for the MOBY API
//
// Version 4.40 / 14. June 2002
//
// Include this header file in any MOBY application
// Please make sure that the include path is set correctly


#ifndef MOBYAPIDEFINED
#define MOBYAPIDEFINED


////////////////////////////////////////////////////////////////////////////
// Includes

#include <3964r.h>


////////////////////////////////////////////////////////////////////////////
// Im- and Export definitions for prototypes

#ifdef DLLROUTINE
#undef DLLROUTINE
#endif

#ifdef DLLDECL
#undef DLLDECL
#endif

#ifdef __MOBY_DLL_IMPL
#define DLLROUTINE __declspec(dllexport)
#else
#define DLLROUTINE __declspec(dllimport)
#endif

#define DLLDECL WINAPI




////////////////////////////////////////////////////////////////////////////
// Constants

#define MAXCHANNEL 4


////////////////////////////////////////////////////////////////////////////
// Type definitions
```

```
//.........................................................
// general definitions

typedef unsigned int  mobyHandle_t;
typedef unsigned int  uInt;
typedef int           mobyType_t;


//.........................................................
// error and status type

typedef struct mobyStatus_d
{
       unsigned int   errorCode      : 5;
       unsigned int   eccDone        : 1;
       unsigned int   diagBatMDS507  : 1;
       unsigned int   batMDS         : 1;
       unsigned int   anwMDS         : 1;
       unsigned int   busyASM        : 1;
       unsigned int   dummy          : 21;
       unsigned int   error          : 1;
} mobyStatus_t;

typedef struct mobyStatusU_d
{
       mobyStatus_t          status;
       unsigned char         s_info;
       unsigned char         hw_type;
       unsigned short int    hw_ver;
       unsigned short int    boot_ver;
       unsigned char         fw_type;
       unsigned short int    fw_ver;
       unsigned char         drv_type;
       unsigned short int    drv_ver;
       unsigned char         interf;
       unsigned char         baud;
       unsigned char         dili;
       unsigned char         mtag;
       unsigned char         fcon;
       unsigned char         ftim;
       unsigned char         sema;
       unsigned char         ant;
       unsigned char         standby;
       unsigned char         anw;
} mobyStatusU_t;


typedef struct mobyStatusMDS_d
{
       mobyStatus_t          status;
       unsigned long int     mds_no;
       unsigned char         mds_type;
       unsigned long int     strz;
       unsigned short int    ssmz;
       unsigned short int    mcod;
       unsigned short int    rbld;
       unsigned char         sleep_time;
} mobyStatusMDS_t;

typedef union mobyErr_d
{
       long          error;
       mobyStatus_t  status;
} mobyErr_t;
```

```
//.............................................................
// definitions for moby_diagnose (for MOBY U only)

#define MOBY_U_MAXFUNC 33

typedef struct funcDesc_d
{
       unsigned char  data[7];
} funcDesc_t;

typedef struct mobyDiagnoseCall_d
{
       mobyStatus_t   status;
       unsigned int   num;
       funcDesc_t     functions[MOBY_U_MAXFUNC];
} mobyDiagnoseCall_t;


#define MOBY_U_MAXERR 233

typedef unsigned char errDesc_t;

typedef struct mobyDiagnoseErr_d
{
       mobyStatus_t   status;
       unsigned int   num;
       errDesc_t      error[MOBY_U_MAXERR];
} mobyDiagnoseErr_t;

#define MOBY_U_MAXMDS 24

typedef struct mdsDesc_d
{
       unsigned char data[4];
} mdsDesc_t;

typedef struct mobyDiagnoseMDS_d
{
       mobyStatus_t   status;
       unsigned int   num;
       mdsDesc_t      mds[MOBY_U_MAXMDS];
} mobyDiagnoseMDS_t;

typedef struct mobyDiagnoseRepeat_d
{
       unsigned int   num;
} mobyDiagnoseRepeat_t;

typedef union mobyDiagnose_d
{
       mobyDiagnoseCall_t    diagCall;
       mobyDiagnoseErr_t     diagErr;
       mobyDiagnoseMDS_t     diagMDS;
       mobyDiagnoseRepeat_t  diagRepeat;
} mobyDiagnose_t;


//.............................................................
// parameters for MOBY I (without multi-channel support)

typedef struct mobyReset_d
{
       // information to set / read LEDs
       BOOL red, green, TxD, RxD, setLED;

       // information to set / read driver
       BOOL dialogOn, moreEC, timeout, eeprom, anwControl, scanFlag, mobyV_On,
       setTreiber;

       // bit field for the DIL switches
       unsigned char dilSwitch;
} mobyReset_t;

typedef struct mobyInterval_d
{
       short int      timebase;
       short int      timeval;
       BOOL           dialogOn;
       BOOL           mobyV_On;
```

```
} mobyInterval_t;


//.....................................................
// parameters for MOBY (with multi-channel support)

typedef struct mobyChannelasm_abta_f_d
{
       unsigned char  timeval        : 6;
       unsigned char  timebase       : 2;
} mobyChannelasm_abta_f_t;

typedef union moby_channelasm_abta_d
{
       unsigned char                 raw;
       mobyChannelasm_abta_f_t       fields;
} mobyChannelasm_abta_t;


typedef struct mobyChannelasm_param_f_d
{
       unsigned char  mode           : 4;
       unsigned char  res            : 1;
       unsigned char  anw            : 3;
} mobyChannelasm_param_f_t;

typedef union mobyChannelasm_param_d
{
       unsigned char                 raw;
       mobyChannelasm_param_f_t      fields;
} mobyChannelasm_param_t;


typedef struct mobyChannelasm_opt_f_d
{
       unsigned char  res1           : 1;
       unsigned char  clear_led      : 1;
       unsigned char  timeout        : 1;
       unsigned char  tst_on         : 1;
       unsigned char  res2           : 1;
       unsigned char  res3           : 1;
       unsigned char  res4           : 1;
       unsigned char  res5           : 1;
} mobyChannelasm_opt_f_t;

typedef union mobyChannelasm_opt_d
{
       unsigned char                 raw;
       mobyChannelasm_opt_f_t        fields;
}  mobyChannelasm_opt_t;


typedef struct mobyChannelasm_d
{
       mobyChannelasm_abta_t         abta;
       mobyChannelasm_param_t        param;
       mobyChannelasm_opt_t          opt;
} mobyChannelasm_t;
```

```
typedef struct mobyUreset_d
{
       unsigned char  standby;
       unsigned char  param;
       unsigned char  option1;
       unsigned char  dili;
       unsigned short mtag;
       unsigned char  fcon;
       unsigned char  ftim;
} mobyUreset_t;

typedef union mobyParameters_d
{
       mobyChannelasm_t      channelasm;
       mobyReset_t           extended;
       mobyInterval_t        interval;
       mobyUreset_t          Ureset;
} mobyParameters_t;


//.............................................................
// structure for multitag GET

#define MOBY_MTGET_MAXMDS   12
#define MOBY_MTGET_MAXDATA 250

typedef struct mobyMtget_data_d
{
       uInt mds;
       unsigned char data[MOBY_MTGET_MAXDATA];
} mobyMtget_data_t;

typedef struct mobyMtget_d
{
       int numMds;
       mobyMtget_data_t mobyMtget_data[MOBY_MTGET_MAXMDS];
} mobyMtget_t;


//.............................................................
// type for accessing DE/DA (when available)

typedef struct mobyDEDA_d
{
       BOOL   bitDA0,bitDA1,bitDA2,bitDA3,bitDE0,bitDE1,bitDE2,bitDE3;
} mobyDEDA_t;

typedef void   (CALLBACK *moby_UnexpCallback_t)     (unsigned char ch1,
                                                     unsigned char ch2,
                                                     unsigned char ch3, int laenge);
typedef void   (CALLBACK *moby_AnwCallback_t)       (unsigned char anwstatus);


////////////////////////////////////////////////////////////////////////////
// Interface of the MOBY API

#if defined(__cplusplus)
extern "C"
{
#endif

DLLROUTINE mobyErr_t DLLDECL moby_open       (LPCSTR com_name, int channel,
                                              mobyHandle_t *handle);
DLLROUTINE mobyErr_t DLLDECL moby_close      (mobyHandle_t handle);
DLLROUTINE mobyErr_t DLLDECL moby_start      (mobyHandle_t handle, mobyType_t type,
                                              BOOL eccOn, mobyParameters_t *param,
                                              HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_stop       (mobyHandle_t handle);
DLLROUTINE mobyErr_t DLLDECL moby_next       (mobyHandle_t handle, HANDLE sync,
                                              mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_status     (mobyHandle_t handle, mobyStatus_t *stat,
                                              HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_statusU    (mobyHandle_t handle, mobyStatusU_t *stat,
                                              HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_diagnose   (mobyHandle_t handle, unsigned char mode,
                                              mobyDiagnose_t *diagnose, HANDLE sync,
                                              mobyErr_t *err);
```

```
DLLROUTINE mobyErr_t DLLDECL moby_read       (mobyHandle_t handle, uInt mdsAddress,
                                              unsigned char *data, uInt length,
                                              HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_s_read     (mobyHandle_t handle, uInt *idData,
                                              uInt mdsAddress, unsigned char *data,
                                              uInt length, HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_getID      (mobyHandle_t handle, unsigned char *idData,
                                              uInt idLength, HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_s_getID    (mobyHandle_t handle, mobyMtget_t *getInfo,
                                              int address, uInt length, HANDLE sync,
                                              mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_write      (mobyHandle_t handle, uInt mdsAddress,
                                              unsigned char *data, uInt length,
                                              HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_s_write    (mobyHandle_t handle, uInt *idData,
                                              uInt mdsAddress, unsigned char *data,
                                              uInt length, HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_init       (mobyHandle_t handle, unsigned char setVal,
                                              uInt mdsLength, HANDLE sync,
                                              mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_s_init     (mobyHandle_t handle, uInt *idData,
                                              unsigned char setVal, uInt mdsLength,
                                              HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_readDE     (mobyHandle_t handle, mobyDEDA_t *deda,
                                              HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_writeDA    (mobyHandle_t handle, mobyDEDA_t *deda,
                                              HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_unexpect   (mobyHandle_t handle,
                                              moby_UnexpCallback_t cbroutine);
DLLROUTINE mobyErr_t DLLDECL moby_anw        (mobyHandle_t handle,
                                              moby_AnwCallback_t cbroutine);
DLLROUTINE mobyErr_t DLLDECL moby_version    (int *major, int *minor);
DLLROUTINE mobyErr_t DLLDECL moby_end        (mobyHandle_t handle, unsigned char mode,
                                              HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_s_end      (mobyHandle_t handle, uInt *idData,
                                              unsigned char mode, HANDLE sync,
                                              mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_repeat     (mobyHandle_t handle, unsigned char mode,
                                              HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_setANT     (mobyHandle_t handle, unsigned char mode,
                                              HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_statusMDS  (mobyHandle_t handle,
                                              mobyStatusMDS_t *statusMDS,
                                              unsigned char mode, unsigned char cweek,
                                              unsigned char year, HANDLE sync,
                                              mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_s_statusMDS (mobyHandle_t handle, uInt *idData,
                                              mobyStatusMDS_t *statusMDS,
                                              unsigned char mode, unsigned char cweek,
                                              unsigned char year, HANDLE sync,
                                              mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_writeOTP   (mobyHandle_t handle, unsigned char *data,
                                              HANDLE sync, mobyErr_t *err);

DLLROUTINE mobyErr_t DLLDECL moby_s_writeOTP (mobyHandle_t handle, uInt *idData,
                                              unsigned char *data, HANDLE sync,
                                              mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_readOTP    (mobyHandle_t handle, unsigned char *data,
                                              HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_s_readOTP  (mobyHandle_t handle, uInt *idData,
                                              unsigned char *data, HANDLE sync,
                                              mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_s_copy     (mobyHandle_t handle, uInt idData1,
                                              int addr1, uInt idData2, int addr2,
                                              uInt len, HANDLE sync, mobyErr_t *err);

#if defined(__cplusplus)
}
#endif


////////////////////////////////////////////////////////////////////////////
// Moby channels

#define MOBY_CHANNEL1         1
#define MOBY_CHANNEL2         2
#define MOBY_CHANNEL3         3
#define MOBY_CHANNEL4         4
```

```
#define MOBY_NOCHANNEL       0


////////////////////////////////////////////////////////////////////////////////
// Moby types

#define MOBY_I        0
#define MOBY_Ia       0      /* MOBY I with intervall reset */
#define MOBY_Ib       1      /* MOBY I with extended reset */
#define MOBY_F        2
#define MOBY_E        3
#define MOBY_L        4
#define MOBY_Ua       5      /* MOBY U with small set of parameters */
#define MOBY_Ub       6      /* MOBY U with large set of parameters */
#define MOBY_Uc       7      /* MOBY U with multitag support */


////////////////////////////////////////////////////////////////////////////////
// Constants for specific moby types

#define MOBY_CHANNEL_PARAM_RESET {{{{0}},{{0}},{{0}}}}


//.............................................................
// working modes for channeled ASMs

// general

#define MOBY_CHANNEL_ALL_RESETPARAM_MODE_IGNORE           0x0    // all ASMs


// for MOBY F

#define MOBY_CHANNEL_F_RESETPARAM_MODE_MDS_F1xx           0xA    // ASM 424/824
#define MOBY_CHANNEL_F_RESETPARAM_MODE_MDS_F4xx           0xB    // ASM 424/824
#define MOBY_CHANNEL_F_RESETPARAM_MODE_MDS_F2xx           0xC    // ASM 424


// for MOBY I

#define MOBY_CHANNEL_I_RESETPARAM_MODE                    0x1    // ASM 424
#define MOBY_CHANNEL_I_RESETPARAM_MODE_MDS507             0x4    // ASM 424


// for MOBY V

#define MOBY_CHANNEL_V_RESETPARAM_MODE                    0x9    // ASM 424


// for MOBY E

#define MOBY_CHANNEL_E_RESETPARAM_MODE                    0x1    // ASM 724


// for MOBY U

#define MOBY_U_DIAG_LASTCALL                              0x02
#define MOBY_U_DIAG_LASTERR                               0x03
#define MOBY_U_DIAG_LASTMDS                               0x04
#define MOBY_U_DIAG_LASTREPEAT                            0x05


//.............................................................
// ANW control for channeled ASMs

#define MOBY_CHANNEL_RESET_PARAM_ANW_NO                   0x0
#define MOBY_CHANNEL_RESET_PARAM_ANW_DETECT               0x1
#define MOBY_CHANNEL_RESET_PARAM_ANW_CONTROL              0x2


////////////////////////////////////////////////////////////////////////////////
// Error and status numbers

#define MOBY_DLL_FEHLER             0x80000000
#define SET_MOBY_ERROR(block,fehler) (MOBY_DLL_FEHLER | block | fehler)

#define MOBY_OK                     0x0000
```

```
#define MOBY_ERRB_API               0x10000
#define MOBY_ERRB_TREIBER           0x20000
#define MOBY_ERRB_ASM               0x30000


#define ERR_MACRO(block,nummer)     ((long) (MOBY_DLL_FEHLER | block | nummer))

// API error numbers

#define MOBY_ERR_FATAL              ERR_MACRO(MOBY_ERRB_API, 0)
#define MOBY_ERR_HANDLE             ERR_MACRO(MOBY_ERRB_API, 1)
#define MOBY_ERR_PARAM              ERR_MACRO(MOBY_ERRB_API, 2)
#define MOBY_ERR_RESPONSE           ERR_MACRO(MOBY_ERRB_API, 3)
#define MOBY_ERR_LAENGE             ERR_MACRO(MOBY_ERRB_API, 4)
#define MOBY_ERR_FULL               ERR_MACRO(MOBY_ERRB_API, 5)
#define MOBY_ERR_MDSTYP             ERR_MACRO(MOBY_ERRB_API, 6)
#define MOBY_ERR_NOHANDLE           ERR_MACRO(MOBY_ERRB_API, 7)
#define MOBY_ERR_SYSTEM             ERR_MACRO(MOBY_ERRB_API, 8)
#define MOBY_ERR_ABORT              ERR_MACRO(MOBY_ERRB_API, 9)
#define MOBY_ERR_NOID               ERR_MACRO(MOBY_ERRB_API,10)
#define MOBY_ERR_STILLOPEN          ERR_MACRO(MOBY_ERRB_API,11)
#define MOBY_ERR_IMPLABORT          ERR_MACRO(MOBY_ERRB_API,12)
#define MOBY_ERR_ALREADYOPEN        ERR_MACRO(MOBY_ERRB_API,13)
#define MOBY_ERR_STARTED            ERR_MACRO(MOBY_ERRB_API,14)
#define MOBY_ERR_NOTSTARTED         ERR_MACRO(MOBY_ERRB_API,15)
#define MOBY_ERR_STATUSPENDING      ERR_MACRO(MOBY_ERRB_API,16)
#define MOBY_ERR_NOTSUPPORTED       ERR_MACRO(MOBY_ERRB_API,17)


#endif  /* MOBYAPIDEFINED */
```

4.2 Headerdatei 3964R.H

```
// header file for the 3964R/Lauf driver
//
// Version 4.40 / 14. June 2002
//
// This header file has to be included into any program source code,
// that intends to use the driver API

//////////////////////////////////////////////////////////////////////////
// definition of the export and import interface

#ifdef __3964R_TREIBER_IMPL
#define DLLROUTINE __declspec(dllexport)
#else
#define DLLROUTINE __declspec(dllimport)
#endif

#define DLLDECL __cdecl

/*
#ifdef __cplusplus
#define DLLROUTINE extern "C"
#else
#define DLLROUTINE extern
#endif
*/

#include "windows.h"


//////////////////////////////////////////////////////////////////////////
// type definitions

typedef signed short int comInt;
typedef comInt comHandle_t;

typedef struct _devConfig_t
{
       int baud,databit,paritaet,stopbit;
       int prot;
       int sswh,swh,zt,qt,to,uew;
       int empf,send;
       BOOL  drop;
       int priority;
} devConfig_t;

typedef devConfig_t *devConfig_p;

typedef void (CALLBACK *comNotifCall)(int event, int status, comHandle_t handle,
int userID);

//////////////////////////////////////////////////////////////////////////
// interface for the 3964R driver
// mostly adopted from the ECCOM interface (available for Win 3.1)

#if defined(__cplusplus)
extern "C"
{
#endif

DLLROUTINE comInt DLLDECL ComOpen           (LPCSTR com_name, int read_number,
                                             int write_number, HWND hwnd);
DLLROUTINE comInt DLLDECL ComRead           (comHandle_t com_handle,
                                             void FAR *read_data, int read_number,
                                             long option);
DLLROUTINE comInt DLLDECL ComWrite          (comHandle_t com_handle,
                                             void FAR *write_data, int write_number,
                                             long option);
DLLROUTINE comInt DLLDECL ComEnableEvent    (comHandle_t com_handle, int com_event,
                                             int user_id, int msg);
DLLROUTINE comInt DLLDECL ComDisableEvent   (comHandle_t com_handle, int com_event);
DLLROUTINE comInt DLLDECL ComGetNotify      (WPARAM wParam, LPARAM lParam,
                                             int FAR *user_id_ptr, int FAR *event_ptr,
                                             int FAR *state_ptr, int FAR *handle_ptr);
```

```
DLLROUTINE comInt DLLDECL ComClose          (comHandle_t com_handle);
DLLROUTINE comInt DLLDECL ComGetVersion     (char FAR *ver_string);
DLLROUTINE comInt DLLDECL ComString         (char FAR *errs, comInt error, comInt typ);
DLLROUTINE comInt DLLDECL ComGetReadState   (comHandle_t com_handle);
DLLROUTINE comInt DLLDECL ComGetWriteState  (comHandle_t com_handle);

// New extension to the API (beyond the old ECCOM interface)
//   -> configuration
//   -> option to use callback based event handler instead of Windows messages

DLLROUTINE comInt DLLDECL ComReadConfig     (LPCSTR devName, devConfig_p conf);
DLLROUTINE comInt DLLDECL ComWriteConfig    (LPCSTR devName, devConfig_p conf,
                                            BOOL force);
DLLROUTINE comInt DLLDECL ComSetNotification(comHandle_t com_handle,
                                            comNotifCall p_callback, int userID);

#if defined(__cplusplus)
}
#endif


// comment:
//
// This interface is adopted from the ECCOM driver package available for
// Windows 3.1. However, the calling semantics have changed slightly for
// some calls. This is necessarry to accomodate the modified driver
// specification.


////////////////////////////////////////////////////////////////////////////
// Error and status codes (according to ECCOM)

#define COM_OK                      0x0000

#define COM_ST_FREE                 0x0000
#define COM_ST_BUSY                 0x0001
#define COM_ST_SUCCESS              0x0003

#define COM_ST_ERROR                0x8000
#define COM_ST_2MANY                0x8001
#define COM_ST_NO_CON               0x8002
#define COM_ST_NO_TRA               0x8004
#define COM_ST_2SMALL               0x8008
#define COM_ST_BCCERR               0x8010
#define COM_ST_TIMCON               0x8020
#define COM_ST_TIMTRA               0x8040
#define COM_ST_TIMQUI               0x8080
#define COM_ST_SCC_BR               0x8100
#define COM_ST_SCC_PY               0x8200
#define COM_ST_SCC_FR               0x8400
#define COM_ST_SCC_OR               0x8800
#define COM_ST_SNDRCV               0x9000
#define COM_ST_SYSERR               0xC000

#define COM_DLL_ERROR               0xA000
#define COM_HANDLE_FALSE            0xA002
#define COM_NO_MEMORY               0xA003
#define COM_2MANY                   0xA004
#define COM_2SMALL                  0xA005
#define COM_DOS_ERROR               0xA006
#define COM_ALREADY_OPEN            0xA007
#define COM_NOT_OPEN                0xA008
#define COM_NO_TIMER                0xA009
#define COM_ERROR_WRITE_OLD         0xA00A
#define COM_COM_BUSY                0xA00B
#define COM_ERROR_POSTMESSAGE       0xA00C
#define COM_CLOSE_ERROR             0xA00D
#define COM_FREE_ERROR              0xA00E
#define COM_CLFR_ERROR              0xA00F
#define COM_UNKNOWN_ID              0xA010
#define COM_UNKNOWN_EVENT           0xA011
#define COM_OPEN_ERROR              0xA012
#define COM_READ_ERROR              0xA013
#define COM_WRITE_ERROR             0xA014
#define COM_SNR_ERROR               0xA015
#define COM_CNF_ERROR               0xA016
#define COM_WRITE_NAK               0xA017
#define COM_WRITE_WRONG             0xA018
```

```
#define COM_DDFINI_ERROR            0xA064
#define COM_FRDPAR_ERROR            0xA065
#define COM_FOPPAR_ERROR            0xA066
#define COM_FRDINI_ERROR            0xA067
#define COM_STRINI_ERROR            0xA068
#define COM_OPENSTR_ERROR           0xA069

#define COM_NO_CONFIG               0xE001
#define COM_REGISTRY                0xE002
#define COM_NO_HANDLE               0xE003

#define COM_ERROR                   0xFFFF


////////////////////////////////////////////////////////////////////////////
// further constants

#define COM1                        "COM1"
#define COM2                        "COM2"
#define COM3                        "COM3"
#define COM4                        "COM4"

#define COM_NO_OPTION               0L

#define COM_OPEN_STD_BUF            -1

#define COM_GET_EVENT               0x0000
#define COM_READ_EVENT              0x0001
#define COM_WRITE_EVENT             0x0002

#define COM_STR                     0
#define COM_STR_OPEN                1
#define COM_STR_RDWR                2
#define COM_STR_STATE               3
#define COM_STR_EVENT               4
#define COM_STR_VERSION             5
#define COM_STR_CLOSE               6
```

4.3 Headerdatei MOBY_API_T.H

```
// Headerfile for the MOBY API TCP
//
// Version 1.00 / 22. August 2003
//
// Include this header file in any MOBY application
// Please make sure that the include path is set correctly


#ifndef MOBY_API_T_H//MOBYAPIDEFINED
#define MOBY_API_T_H//MOBYAPIDEFINED


////////////////////////////////////////////////////////////////////////////
// Includes



#include <windows.h>
//#include <winsock.h>

////////////////////////////////////////////////////////////////////////////
// Im- and Export definitions for prototypes

#ifdef DLLROUTINE
#undef DLLROUTINE
#endif

#ifdef DLLDECL
#undef DLLDECL
#endif

#ifdef __MOBY_DLL_IMPL
#define DLLROUTINE __declspec(dllexport)
#else
#define DLLROUTINE __declspec(dllimport)
#endif

#define DLLDECL WINAPI

////////////////////////////////////////////////////////////////////////////
// Type definitions

//.........................................................
// general definitions

typedef unsigned int  mobyHandle_t;
typedef unsigned int  uInt;
typedef int           mobyType_t;


//.........................................................
// error and status type

typedef struct mobyStatus_d
{
       unsigned int   errorCode      : 5;
       unsigned int   eccDone        : 1;
       unsigned int   diagBatMDS507  : 1;
       unsigned int   batMDS         : 1;
       unsigned int   anwMDS         : 1;
       unsigned int   busyASM        : 1;
       unsigned int   dummy          : 21;
       unsigned int   error          : 1;
} mobyStatus_t;

typedef struct mobyStatusU_d
{
       mobyStatus_t          status;
       unsigned char         s_info;
       unsigned char         hw_type;
       unsigned short int    hw_ver;
       unsigned short int    boot_ver;
       unsigned char         fw_type;
```

```
        unsigned short int    fw_ver;
        unsigned char         drv_type;
        unsigned short int    drv_ver;
        unsigned char         interf;
        unsigned char         baud;
        unsigned char         dili;
        unsigned char         mtag;
        unsigned char         fcon;
        unsigned char         ftim;
        unsigned char         sema;
        unsigned char         ant;
        unsigned char         standby;
        unsigned char         anw;
} mobyStatusU_t;


typedef struct mobyStatusMDS_d
{
        mobyStatus_t          status;
        unsigned long int     mds_no;
        unsigned char         mds_type;
        unsigned long int     strz;
        unsigned short int    ssmz;
        unsigned short int    mcod;
        unsigned short int    rbld;
        unsigned char         sleep_time;
} mobyStatusMDS_t;

typedef union mobyErr_d
{
        long           error;
        mobyStatus_t   status;
} mobyErr_t;


//.............................................................
// definitions for moby_diagnose (for MOBY U only)

#define MOBY_U_MAXFUNC 33

typedef struct funcDesc_d
{
        unsigned char data[7];
} funcDesc_t;

typedef struct mobyDiagnoseCall_d
{
        mobyStatus_t   status;
        unsigned int   num;
        funcDesc_t     functions[MOBY_U_MAXFUNC];
} mobyDiagnoseCall_t;


#define MOBY_U_MAXERR 233

typedef unsigned char errDesc_t;

typedef struct mobyDiagnoseErr_d
{
        mobyStatus_t   status;
        unsigned int   num;
        errDesc_t      error[MOBY_U_MAXERR];
} mobyDiagnoseErr_t;

#define MOBY_U_MAXMDS 24

typedef struct mdsDesc_d
{
        unsigned char data[4];
} mdsDesc_t;

typedef struct mobyDiagnoseMDS_d
{
        mobyStatus_t   status;
        unsigned int   num;
        mdsDesc_t      mds[MOBY_U_MAXMDS];
} mobyDiagnoseMDS_t;

typedef struct mobyDiagnoseRepeat_d
```

```
{
       unsigned int   num;
} mobyDiagnoseRepeat_t;

typedef union mobyDiagnose_d
{
       mobyDiagnoseCall_t    diagCall;
       mobyDiagnoseErr_t     diagErr;
       mobyDiagnoseMDS_t     diagMDS;
       mobyDiagnoseRepeat_t  diagRepeat;
} mobyDiagnose_t;



typedef struct mobyUreset_d
{
       unsigned char  standby;
       unsigned char  param;
       unsigned char  option1;
       unsigned char  dili;
       unsigned short mtag;
       unsigned char  fcon;
       unsigned char  ftim;
} mobyUreset_t;

typedef union mobyParameters_d
{
       mobyUreset_t   Ureset;
} mobyParameters_t;


//.............................................................
// structure for multitag GET

#define MOBY_MTGET_MAXMDS   12
#define MOBY_MTGET_MAXDATA 250

typedef struct mobyMtget_data_d
{
       uInt mds;
       unsigned char data[MOBY_MTGET_MAXDATA];
} mobyMtget_data_t;

typedef struct mobyMtget_d
{
       int numMds;
       mobyMtget_data_t mobyMtget_data[MOBY_MTGET_MAXMDS];
} mobyMtget_t;


typedef void (CALLBACK *moby_UnexpCallback_t)       (unsigned char ch1,
                                                    unsigned char ch2,
                                                    unsigned char ch3, int laenge);
typedef void (CALLBACK *moby_AnwCallback_t)         (unsigned char anwstatus);



////////////////////////////////////////////////////////////////////////
// Interface of the MOBY API

#if defined(__cplusplus)
extern "C"
{
#endif

DLLROUTINE mobyErr_t DLLDECL moby_open              (char *comStr, int channel,
                                                    mobyHandle_t *handle,
                                                    unsigned short sPort);
DLLROUTINE mobyErr_t DLLDECL moby_close             (mobyHandle_t handle);
DLLROUTINE mobyErr_t DLLDECL moby_start             (mobyHandle_t handle,
                                                    mobyType_t type, BOOL eccOn,
                                                    mobyParameters_t *param,
                                                    HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_stop              (mobyHandle_t handle);
DLLROUTINE mobyErr_t DLLDECL moby_statusU           (mobyHandle_t handle,
                                                    mobyStatusU_t *stat,
                                                    HANDLE sync, mobyErr_t *err);
```

```
DLLROUTINE mobyErr_t DLLDECL moby_diagnose          (mobyHandle_t handle,
                                                    unsigned char mode,
                                                    mobyDiagnose_t *diagnose,
                                                    HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_s_read            (mobyHandle_t handle, uInt *idData,
                                                    uInt mdsAddress,
                                                    unsigned char *data, uInt length,
                                                    HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_s_getID           (mobyHandle_t handle,
                                                    mobyMtget_t *getInfo, int address,
                                                    uInt length, HANDLE sync,
                                                    mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_s_write           (mobyHandle_t handle, uInt *idData,
                                                    uInt mdsAddress,
                                                    unsigned char *data, uInt length,
                                                    HANDLE sync, mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_s_init            (mobyHandle_t handle, uInt *idData,
                                                    unsigned char setVal,
                                                    uInt mdsLength, HANDLE sync,
                                                    mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_unexpect          (mobyHandle_t handle,
                                                    moby_UnexpCallback_t cbroutine);
DLLROUTINE mobyErr_t DLLDECL moby_anw               (mobyHandle_t handle,
                                                    moby_AnwCallback_t cbroutine);
DLLROUTINE mobyErr_t DLLDECL moby_version           (int *major, int *minor);
DLLROUTINE mobyErr_t DLLDECL moby_s_end             (mobyHandle_t handle, uInt *idData,
                                                    unsigned char mode, HANDLE sync,
                                                    mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_setANT            (mobyHandle_t handle,
                                                    unsigned char mode, HANDLE sync,
                                                    mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_s_statusMDS       (mobyHandle_t handle, uInt *idData,
                                                    mobyStatusMDS_t *statusMDS,
                                                    unsigned char mode,
                                                    unsigned char cweek,
                                                    unsigned char year, HANDLE sync,
                                                    mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_s_writeOTP        (mobyHandle_t handle, uInt *idData,
                                                    unsigned char *data, HANDLE sync,
                                                    mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_s_readOTP         (mobyHandle_t handle, uInt *idData,
                                                    unsigned char *data, HANDLE sync,
                                                    mobyErr_t *err);
DLLROUTINE mobyErr_t DLLDECL moby_s_copy            (mobyHandle_t handle, uInt idData1,
                                                    int addr1, uInt idData2, int addr2,
                                                    uInt len, HANDLE sync,
                                                    mobyErr_t *err);

#if defined(__cplusplus)
}
#endif

////////////////////////////////////////////////////////////////////////////////
// Moby channels

#define MOBY_CHANNEL1 1
#define MOBY_CHANNEL2 2
#define MOBY_CHANNEL3 3
#define MOBY_CHANNEL4 4

#define MOBY_NOCHANNEL 0


////////////////////////////////////////////////////////////////////////////////
// Moby types

#define MOBY_Ua     5   /* MOBY U with small set of parameters */
#define MOBY_Ub     6   /* MOBY U with large set of parameters */
#define MOBY_Uc     7   /* MOBY U with multitag support */


////////////////////////////////////////////////////////////////////////////////
// Constants for specific moby types

#define MOBY_CHANNEL_PARAM_RESET {{{{0}},{{0}},{{0}}}}


//.............................................................
// working modes for channeled ASMs
```

```
// general

#define MOBY_CHANNEL_ALL_RESETPARAM_MODE_IGNORE           0x0       // all ASMs


// for MOBY U

#define MOBY_U_DIAG_LASTCALL         0x02
#define MOBY_U_DIAG_LASTERR          0x03
#define MOBY_U_DIAG_LASTMDS          0x04
#define MOBY_U_DIAG_LASTREPEAT       0x05


//.............................................................
// ANW control for channeled ASMs


////////////////////////////////////////////////////////////////////////////////
// Error and status numbers

#define MOBY_DLL_FEHLER              0x80000000
#define SET_MOBY_ERROR(block,fehler) (MOBY_DLL_FEHLER | block | fehler)

#define MOBY_OK                      0x0000

#define MOBY_ERRB_API                0x10000
#define MOBY_ERRB_TREIBER            0x20000
#define MOBY_ERRB_ASM                0x30000


#define ERR_MACRO(block,nummer)      ((long) (MOBY_DLL_FEHLER | block | nummer))

// API error numbers

#define MOBY_ERR_FATAL               ERR_MACRO(MOBY_ERRB_API, 0)
#define MOBY_ERR_HANDLE              ERR_MACRO(MOBY_ERRB_API, 1)
#define MOBY_ERR_PARAM               ERR_MACRO(MOBY_ERRB_API, 2)
#define MOBY_ERR_RESPONSE            ERR_MACRO(MOBY_ERRB_API, 3)
#define MOBY_ERR_LAENGE              ERR_MACRO(MOBY_ERRB_API, 4)
#define MOBY_ERR_FULL                ERR_MACRO(MOBY_ERRB_API, 5)
#define MOBY_ERR_MDSTYP              ERR_MACRO(MOBY_ERRB_API, 6)
#define MOBY_ERR_NOHANDLE            ERR_MACRO(MOBY_ERRB_API, 7)
#define MOBY_ERR_SYSTEM              ERR_MACRO(MOBY_ERRB_API, 8)
#define MOBY_ERR_ABORT               ERR_MACRO(MOBY_ERRB_API, 9)
#define MOBY_ERR_NOID                ERR_MACRO(MOBY_ERRB_API,10)
#define MOBY_ERR_STILLOPEN           ERR_MACRO(MOBY_ERRB_API,11)
#define MOBY_ERR_IMPLABORT           ERR_MACRO(MOBY_ERRB_API,12)
#define MOBY_ERR_ALREADYOPEN         ERR_MACRO(MOBY_ERRB_API,13)
#define MOBY_ERR_STARTED             ERR_MACRO(MOBY_ERRB_API,14)
#define MOBY_ERR_NOTSTARTED          ERR_MACRO(MOBY_ERRB_API,15)
#define MOBY_ERR_STATUSPENDING       ERR_MACRO(MOBY_ERRB_API,16)
#define MOBY_ERR_NOTSUPPORTED        ERR_MACRO(MOBY_ERRB_API,17)

#define MOBY_ERR_WRONTIPADR          ERR_MACRO(MOBY_ERRB_API,18)
#define MOBY_ERR_INVALIDSOCK         ERR_MACRO(MOBY_ERRB_API,19)
#define MOBY_ERR_CONNECT             ERR_MACRO(MOBY_ERRB_API,20)
#define MOBY_ERR_KENNUNG             ERR_MACRO(MOBY_ERRB_API,21)


#endif  /* MOBYAPIDEFINED */
```

5 Beispielapplikation

Für den leichten Einstieg in die Realisierung einer Anwenderapplikation wird zusätzlich zur C-Bibliothek für jede Ankopplungsvariante (seriell und Ethernet) eine Beispielapplikation im Sourcecode ausgeliefert. Die Beispielapplikationen stehen auch als ablauffähiges Programm zur Verfügung.

5.1 Beispielapplikation als Source für serielle Ankopplung an PC

Die Beispielapplikation als ablauffähiges Programm ist eine Anwendung für SLA 81 an ASM 824 als READ-System mit MDS F4xx. Das ASM 824 muss an der COM 2-Schnittstelle und die SLA 81 am Kanal 4 des ASM 824 betrieben werden.
Darüber hinaus kann der Sourcecode dieser Beispielapplikation als READ-System für SLA 81 an ASM 824 mit MDS F1xx oder SLA 71 an ASM 724 eingesetzt werden, indem die entsprechende Define-Anweisung auszukommentieren ist.

```
// Example code for MOBY API
// Simple open/start/read/stop/close/stop sequence

#include <stdlib.h>
#include <stdio.h>
#include <windows.h>
#include <moby_api.h>

mobyHandle_t comdev1;
typedef DWORD MDS_address;

typedef unsigned char zeichen;


// default test: MOBY F and MDS F4xx

// enable the next line to test with MOBY E
//#define USE_MOBY_E

// enable the next line to test with MOBY F and MDS F1xx
//#define USE_MDS_F1xx


// callback routine for unexpected telegramms
// we don't expect any -> i.e. error -> i.e.  stop communication

void __stdcall unex_cb(unsigned char c1,unsigned char c2, unsigned char c3, int len)
{
       printf("Got unexpected telegramm\n");
       moby_stop(comdev1);
}


// start and synchronize read request

int MOBY_read(int comdev,FAR void* data, MDS_address MDS, int length, void (*(CALLBACK
status(int))) )
{
       HANDLE sync;
       mobyErr_t      err1,err2;

       sync=CreateEvent(NULL,FALSE,FALSE,NULL);

       err1=moby_read(comdev, MDS, (zeichen *) data, length, sync, &err2);
       if (err1.error==MOBY_OK)
       {
              WaitForSingleObject(sync,INFINITE);
              err1.error=(volatile int) err2.error;
       }

       CloseHandle(sync);
```

```
        return err1.error;
}

// start and synchronize reset/start

int MOBY_reset(int comdev, void (*(CALLBACK status(int))) )
{
        HANDLE                sync;
        mobyErr_t             err1,err2;
        mobyParameters_t      param=MOBY_CHANNEL_PARAM_RESET;
                                      // This initialization should be done to avoid
                                      side effects

        sync=CreateEvent(NULL,FALSE,FALSE,NULL);

        // As param has been initialized, only necessary fields have to be set now

#ifdef USE_MOBY_E
        param.channelasm.param.fields.mode=MOBY_CHANNEL_E_RESETPARAM_MODE;
#else
        param.channelasm.param.fields.anw=MOBY_CHANNEL_RESET_PARAM_ANW_DETECT;
#ifdef USE_MDS_F1xx
        param.channelasm.param.fields.mode=MOBY_CHANNEL_F_RESETPARAM_MODE_MDS_F1xx;
#else
        param.channelasm.param.fields.mode=MOBY_CHANNEL_F_RESETPARAM_MODE_MDS_F4xx;
#endif
        param.channelasm.opt.fields.clear_led=1;
#endif

        // change the parameters in the following call to adapt to
        // other MOBY types

#ifdef USE_MOBY_E
        err1=moby_start(comdev, MOBY_E, FALSE, &param, sync, &err2);
#else
        err1=moby_start(comdev, MOBY_F, FALSE, &param, sync, &err2);
#endif

        if (err1.error==MOBY_OK)
        {
                WaitForSingleObject(sync,INFINITE);
                err1.error=(volatile int) err2.error;
        }

        CloseHandle(sync);

        return err1.error;
}


void main()
{
        int err2,major,minor;
        mobyErr_t err,merr2;
        unsigned char buf[300];

        // request the version

        err=moby_version(&major,&minor);

        printf("Moby DLL Version: %i.%i\n",major, minor);

        // open COM interface

        printf("Open\n");

        // adapt the line below to open other interface
        // and to use other channels

        err=moby_open("COM2",MOBY_CHANNEL4,&comdev1);
//      err=moby_open("COM3",MOBY_NOCHANNEL,&comdev1);

        // do reset/start

        printf("done - now doing start after open returned with %x\n",err.error);

        err.error=MOBY_reset(comdev1,NULL);
```

```
        printf("done\n");

        printf("Opened device1: %x with error %x\n",comdev1,err.error);


        // register callback for reset telegramms

        moby_unexpect(comdev1,unex_cb);


        // read

        printf("Read Nr, length %i: ",4);
#ifdef USE_MDS_F1xx
        err2=MOBY_read(comdev1,&buf,0,5,NULL);
#else
#ifdef USE_MOBY_E
        err2=MOBY_read(comdev1,&buf,0,4,NULL);
#else
        err2=MOBY_read(comdev1,&buf,64,4,NULL);
#endif
#endif
        printf("Result : %x - %02x %02x %02x %02x\n",err2,buf[0],buf[1],buf[2],buf[3]);

        // stop

        merr2=moby_stop(comdev1);
        printf("Stop : %x \n",merr2.error);

        // close

        merr2=moby_close(comdev1);
        printf("Close: %x \n",merr2.error);

}
```

5.2 Beispielapplikation als Source für Ankopplung an Ethernet

Für eine einfache Anwendung SLG U92 an ASM 480 als READ-System steht eine Beispielapplikation als Source TEST.CPP (VC++ 6.0) zur Verfügung. Vor der Compilierung der Applikation muss die in TEST.CPP vordefinierte IP-Adresse „157.163.170.2“ der tatsächlichen IP-Adresse angepasst werden.

```
// Example code for MOBY API T
// Simple open/start/read/stop/close/stop sequence

#include <stdlib.h>
#include <stdio.h>
#include <windows.h>
#include "moby_api_t.h"

mobyHandle_t comdev1;
typedef DWORD MDS_address;

typedef unsigned char zeichen;


// callback routine for unexpected telegramms
// we don't expect any -> i.e. error -> i.e. stop communication

void __stdcall unex_cb(unsigned char c1,unsigned char c2, unsigned char c3, int len)
{
       printf("Got unexpected telegramm\n");
       moby_stop(comdev1);
}


// start and synchronize read request

int MOBY_read  (int comdev,FAR void* data, MDS_address MDS, int length,
       void (*(CALLBACK status(int))) )
{
       HANDLE sync;
       mobyErr_t      err1,err2;
       unsigned       int mdsID[2];
       unsigned       int adress=0;
       mdsID[0]       = 0;

       sync = CreateEvent(NULL,FALSE,FALSE,NULL);


       err1 = moby_s_read(comdev,&mdsID[0],adress,(zeichen *) data,length,sync,&err2);
       if(err1.error==MOBY_OK)
       {
              WaitForSingleObject (sync,INFINITE);
              err1.error = (volatile int) err2.error;
       }

       CloseHandle (sync);

       return err1.error;
}


// start and synchronize reset/start

int MOBY_reset(int comdev, void (*(CALLBACK status(int))) )
{
       HANDLE sync;
       mobyErr_t      err1,err2;

       mobyParameters_t      ResetParam;
       unsigned       int moby_str;
                      // This initialization should be done to avoid side effects

       sync = CreateEvent(NULL,FALSE,FALSE,NULL);
```

```
        ResetParam.Ureset.standby     = 0x00;
        ResetParam.Ureset.param       = 0x26;
        ResetParam.Ureset.option1     = 0x00;
        ResetParam.Ureset.dili        = 0x05;
        ResetParam.Ureset.mtag        = 0x0002;
        ResetParam.Ureset.fcon        = 0x00;
        ResetParam.Ureset.ftim        = 0x00;
        moby_str                      = MOBY_Uc;


        err1 = moby_start(comdev, moby_str, FALSE, &ResetParam, sync, &err2);


        if (err1.error==MOBY_OK)
        {
                WaitForSingleObject (sync,INFINITE);
                err1.error = (volatile int) err2.error;
        }

        CloseHandle(sync);

        return err1.error;
}


void main()
{
        int err2,major,minor;
        mobyErr_t err,merr2;
        unsigned char buf[300];
        unsigned short port_nr = 8000;
        #define IP_ADRESSE "157.163.170.2"

        // request the version

        err = moby_version(&major,&minor);

        printf("Moby DLL Version: %i.%i\n",major, minor);

        // open COM interface

        printf("Open\n");

        // adapt the line beloew to open other interface
        // and to use other channels


        err = moby_open(IP_ADRESSE,0,&comdev1,port_nr);

        // do reset/start

        printf("done - now doing start after open returned with %x\n",err.error);

        err.error = MOBY_reset(comdev1,NULL);

        printf("done\n");

        printf("Opened device1: %x with error %x\n",comdev1,err.error);


        // register callback for reset telegramms

        moby_unexpect(comdev1,unex_cb);


        // read

        printf("Read Nr, length %i: ",4);

        err2 = MOBY_read(comdev1,&buf,64,4,NULL);

        printf("Result : %x - %02x %02x %02x %02x\n",err2,buf[0],buf[1],buf[2],buf[3]);

        // stop

        merr2 = moby_stop(comdev1);
        printf("Stop : %x \n",merr2.error);

        // close
```

```
        merr2 = moby_close(comdev1);
        printf("Close: %x \n",merr2.error);

}
```

A Beschreibung der Kommunikation zum ASM 424/724/824 mit 3964R-Protokoll

Dieser Anhang ist gedacht für Anwender, die ihre Applikation direkt auf Betriebssystem- oder Treiberebene aufsetzen und nicht die C-Bibliothek MOBY API verwenden.

Weitere Informationen zur Treiberebene 3964R finden Sie auch im Benutzerhandbuch „3964R-Protokoll unter Windows NT 4.0/95“ (als PDF-Datei auf der CD „Software MOBY" enthalten).

A.1 Allgemeines

Die Anschaltmodule ASM 424/724/824 arbeiten mit einer Baudrate von 9,6 kBaud, 19,2 kBaud oder 38,4 kBaud, die Erkennung erfolgt automatisch.
Das Datenformat besteht aus 1 Startbit, 8 Datenbits, 1 Paritybit (Parity ergänzt auf insgesamt ungerade Anzahl von Einsen) und 1 Stopbit.
Nach dem Einschalten (Hochlauf) der Baugruppe versucht das ASM ein Hochlauftelegramm abzusetzen, wobei die 3 verschiedenen Baudraten bis zu zweimal durchgetestet werden.
Wurde das Telegramm erfolgreich abgesetzt, so gilt die Baudrate als erkannt.
Im anderen Fall wartet das ASM auf ein Telegramm (STX) und quittiert bei gefundener Baudrate beim zweiten STX mit DLE. Die Baudrate kann während des Betriebs nicht verändert werden.

Achtung

Wird durch einen Übertragungsfehler ein Telegramm nicht abgesetzt, werden alle vier Kanäle zurückgesetzt (die Befehle werden gelöscht).

A.2 Protokolleinstellungen

Bei einem Initialisierungskonflikt ist das ASM fest auf Master parametriert.
Der 3964R-Treiber in dem ASM arbeitet mit folgenden Einstellungen:

- Versuche zum Verbindungsaufbau: 3
- Versuche zur Blockübertragung: 6
- Quittungsverzugszeit: 2 s
- Zeichenverzugszeit: 220 ms
- Blockwartezeit: 10 s
- Wartezeit für zu sendende Folgetelegramme: 10 bis 20 ms

Über den Parametrier-Schalter (8) ist die Schnittstellenart einstellbar:
Oben: RS 422
Unten: RS 232

Die übrigen Parameter werden mittels RESET-Befehl übergeben.

A.3 Anzeigen (LED) auf der 3964R-Schnittstellenseite des ASM

Tabelle A-1 LED-Anzeigen auf dem ASM

LED	Bemerkung
ON (grün)	Betriebspannungsanzeige (Versorgungsspannung am ASM vorhanden)
ACT (grün)	Diese LED blinkt einmal kurz, wenn ein Befehl fertig bearbeitet wurde (nicht beim Statusbefehl).
SF/BF (rot)	• Dauerhaft ein während des Hochlaufs • Aus bei korrekt empfangenem oder gesendetem Telegramm • Blinkt (1 bis 2-mal pro Sekunde) bei einem Sende- oder Empfangsfehler (Telegramm konnte gemäß 3964R-Protokoll nicht übertragen werden).

A.4 Allgemeiner Kommunikationsablauf

Das ASM akzeptiert nur dann einen Befehl, wenn der vorangegangene abgeschlossen ist. Ausgenommen sind RESET- und Statusbefehle, die parallel dazu aufgesetzt werden können (RESET- und Statusbefehle werden sofort bearbeitet).

Achtung

Bei zwei hintereinander abgeschickten Statusbefehlen ohne Quittungsrückmeldung vom ersten Befehl wird immer nur eine Quittung für den letzten Statusbefehl ausgegeben. Beim RESET-Befehl sollte immer die Quittung abgewartet werden, bevor ein neuer Befehl aufgesetzt wird.

A.5 Befehlsübersicht

Tabelle A-2 Liste der Befehle

Befehl	Code	Beschreibung
RESET	00	Rücksetzen sowohl des aktiven Befehls als auch aller unbearbeiteten Befehle im Buffer des jeweiligen Kanals (Befehlsabbruch). Voreinstellung der Betriebsart des ASM. Im Speziellen sind das: • Betrieb mit oder ohne ANW bzw. Anwesenheitssteuerung • MOBY-Betriebsart (E/F oder I)
READ	50	Lesen von Daten aus dem MDS (40: mit ECC)
WRITE	51	Schreiben von Daten auf dem MDS (41: mit ECC)
INIT	18	Initialisierung des Datenspeichers (1A: mit ECC)
STATUS	01	Statusabfrage des ASM • Anwesenheit, Befehl aktiv • Status des ASM

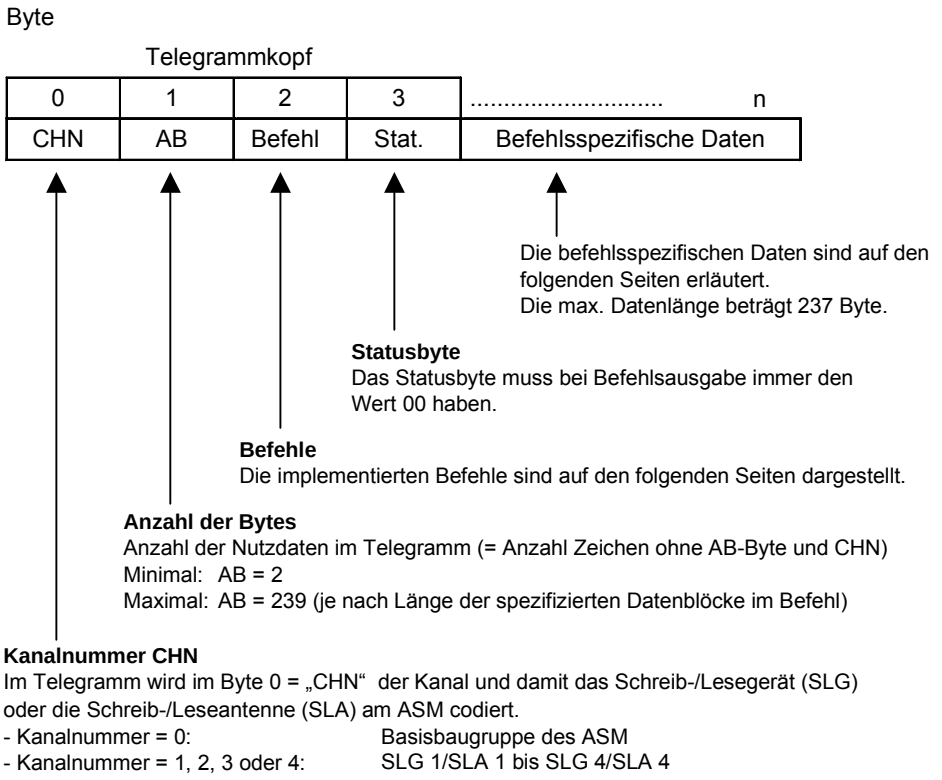
Tabelle A-2 Liste der Befehle

Befehl	Code	Beschreibung
NEXT	07	NEXT-Befehl Die Befehlsbearbeitung des MDS im Feld des SLG wird als beendet erklärt. Der nächste ins Feld kommende MDS wird mit dem nach NEXT gestarteten Befehl bearbeitet (Voraussetzung: Die Anwesenheitssteuerung ist aktiviert).

Achtung

Der ECC-Betrieb ist nur bei MOBY I mit dem ASM 424 möglich.

Allgemeiner Telegrammaufbau



A.6 Telegrammaufbau der Befehle/Quittungen von und zum ASM

Die maximale Telegrammlänge ist auf 241 Byte begrenzt (inkl. Telegrammkopf von 7 Byte).

Datenformat

Soweit nicht anders vermerkt, sind alle Zahlen im Hex-Format (hex) angegeben.

Kanalnummer CHN

Im Telegramm wird im Byte 0 = „CHN“ der Kanal und damit das Schreib-/Lesegerät (SLG) oder die Schreib-/Leseantenne (SLA) am ASM codiert.

- Kanalnummer = 0: Basisbaugruppe des ASM
- Kanalnummer = 1, 2, 3 oder 4: SLG 1/SLA 1 bis SLG 4/SLA 4

A.6.1 Hochlauftelegramm

Byte

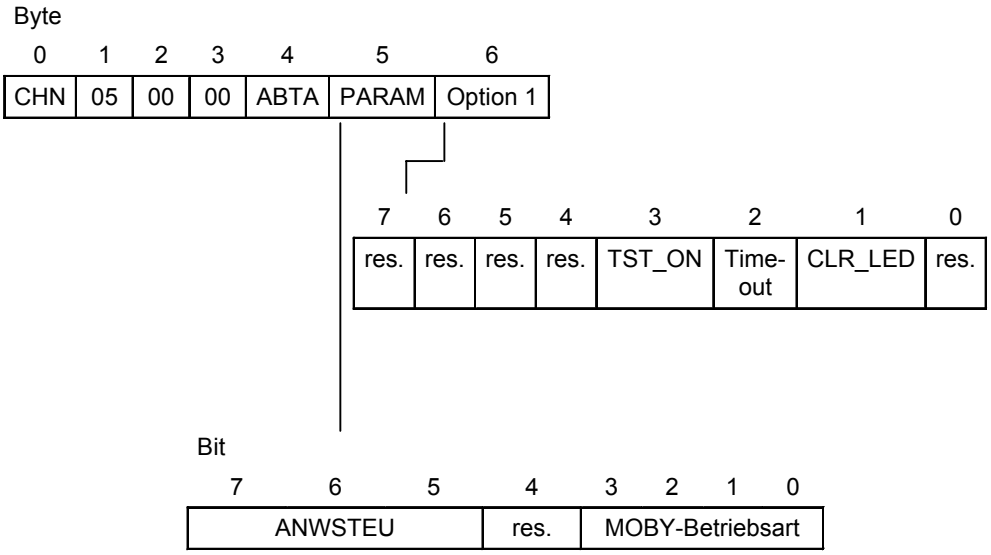
0	1	2	3
CHN	02	00	0F

CHN 00 hex = Basisbaugruppe des ASM

Das Hochlauftelegramm kommt als automatische „Quittung“ nach dem Hardware-Hochlauf. Es wird nicht von den Kanälen, sondern einmal vom Gesamtgerät gesendet, daher ist CHN gleich 0.

A.6.2 RESET

Befehl



CHN 01, 02, 03 oder 04 hex = SLG 1/SLA 1 bis SLG 4/SLA 4
ABTA Einstellung der Abtastzeit bei MOBY I-Long range mit ASM 424
PARAM **ANWSTEU**
000 = Betrieb ohne ANW-Steuerung, ohne ANW-Kontrolle
001 = Betrieb ohne ANW-Steuerung, mit ANW-Kontrolle [1)]
010 = Betrieb mit ANW-Steuerung und ANW-Kontrolle über die Firmware [2)]

1) Betriebsarten mit MDS F1xx (MOBY F, read only) sind grundsätzlich mit ANW-Kontrolle zu parametrieren.

2) Die ANW-Steuerung gilt nur für ASM 424 (für ASM 824 mit MDS F1xx und ASM 724 nicht zugelassen).

MOBY-Betriebsart

Folgende MOBY-Betriebsarten können an dem jeweiligen ASM eingestellt werden.

MOBY I, ASM 424:

1 = MOBY I/MOBY E (SLG 7x)
2 = reserviert
4 = MOBY I mit MDS 507
8 = reserviert für MOBY I-Dialog
9 = reserviert für MOBY V
A = MOBY F: mit MDS F1xx
B = MOBY F: mit MDS F4xx
C = MOBY F: mit MDS F2xx

MOBY E, ASM 724:

0 = Alle im RESET-Telegramm enthaltenen Parameter werden ignoriert. (default)
1 = MOBY E (SLA 71)

MOBY F, ASM 824:

0 = Alle im RESET-Telegramm enthaltenen Parameter werden ignoriert. (default)
A = MOBY F (SLA 81) und MDS F1xx (read only)
B = MOBY F (SLA 81) und MDS F4xx (read/write)

Option 1

TST_ON = 1:

ASM antwortet mit Fehler, wenn Feldstörungen am SLG (nur bei ASM 424).

Timeout = 1:

ASM antwortet bei Befehl mit Fehler, wenn kein MDS im Feld ist (nur bei ASM 424).

CLR_LED = 1:

ERROR_LED (kanalbezogen) wird gelöscht.

Achtung

Die Parameter im RESET-Telegramm werden nur mit dem ersten Telegramm nach Spannung EIN übernommen. D.h., wenn die Parameter geändert werden sollen, muss vorher die Spannung aus und wieder eingeschaltet werden.
Alle Kanäle müssen mit dem gleichen MDS parametriert werden, eine Mischung ist nicht möglich.

Quittung

Byte

0	1	2	3	4	5	6
CHN	05	00	Status	high	low	Res. 1

bzw.

0	1	2	3
CHN	02	00	Status

CHN 01, 02, 03 oder 04 hex = SLG 1/SLA 1 bis SLG 4/SLA 4
Status MOBY-Status
high Firmware-Stand in ASCII
low Firmware-Stand in ASCII
Res.1 reserviert

A.6.3 WRITE

Befehl

Byte

0	1	2	3	4	5	6	ab 7
CHN	AB	51*)	00	ADR H	ADR L	Länge	Daten

*) Mit ECC: Befehl = 41

CHN 01, 02, 03 oder 04 hex = SLG 1/SLA 1 bis SLG 4/SLA 4
AB Anzahl der folgenden Zeichen im Telegramm
ADR H Anfangsadresse auf MDS (höherwertiger Adressteil)
ADR L Anfangsadresse auf MDS (niederwertiger Adressteil)
Länge Länge des Datenblocks (max. 234 Byte)
Daten zu schreibende Daten

Quittung

Byte

0	1	2	3
CHN	02	51*)	Status

*) Mit ECC: Befehl = 41

CHN 01, 02, 03 oder 04 hex = SLG 1/SLA 1 bis SLG 4/SLA 4
Status MOBY-Status

A.6.4 READ

Befehl

Byte

0	1	2	3	4	5	6
CHN	05	50*)	00	ADR H	ADR L	Länge

*) Mit ECC: Befehl = 40

CHN 01, 02, 03 oder 04 hex = SLG 1/SLA 1 bis SLG 4/SLA 4
ADR H Anfangsadresse auf MDS (höherwertiger Adressteil)
ADR L Anfangsadresse auf MDS (niederwertiger Adressteil)
Länge Länge der zu lesenden Daten (max. 234 Byte)

Quittung

Byte

0	1	2	3	4	5	6	7 bis ...
CHN	AB	50*)	Status	ADR H	ADR L	Länge	Daten

*) Mit ECC: Befehl = 40

CHN 01, 02, 03 oder 04 hex = SLG 1/SLA 1 bis SLG 4/SLA 4
AB Anzahl der folgenden Zeichen im Telegramm
Status MOBY-Status
ADR H Anfangsadresse auf MDS (höherwertiger Adressteil)
ADR L Anfangsadresse auf MDS (niederwertiger Adressteil)
Länge Länge des Datenblocks (gelesene Daten)
Daten gelesene Daten

A.6.5 INIT

Befehl

Byte

0	1	2	3	4	5	6	7
CHN	06	18*)	00	INIT-Muster	00	ADR H	ADR L

*) Mit ECC: Befehl = 1A

CHN	01, 02, 03 oder 04 hex = SLG 1/SLA 1 bis SLG 4/SLA 4
INIT-Muster	Beim Initialisieren wird der MDS mit dem Wert „INIT-Muster“ beschrieben.
ADR H	Größe des vorzubesetzenden Speichers (höherwertiger Adressteil)
ADR L	Größe des vorzubesetzenden Speichers (niederwertiger Adressteil)

Quittung

Byte

0	1	2	3
CHN	02	18*)	Status

*) Mit ECC: Befehl = 1A

CHN	01, 02, 03 oder 04 hex = SLG 1/SLA 1 bis SLG 4/SLA 4
Status	MOBY-Status

A.6.6 STATUS

Befehl

Byte

0	1	2	3
CHN	02	01	00

CHN 01, 02, 03 oder 04 hex = SLG 1/SLA 1 bis SLG 4/SLA 4

Quittung

Byte

0	1	2	3	4	5	6	7
CHN	06	01	Status	ANW/Busy	Res.	Res.	Res.

7	6	5	4	3	2	1	0
0	0	0	0	0	0	ANW	Busy

CHN	01, 02, 03 oder 04 hex = SLG 1/SLA 1 bis SLG 4/SLA 4
Status	Anzeige ERR_LED
ANW	Datenträger im Feld
Busy	Befehl in Bearbeitung
Res.	00 hex (Reserve)

A.6.7 NEXT

Befehl

Byte

0	1	2	3
CHN	02	07	00

CHN 01, 02, 03 oder 04 hex = SLG 1/SLA 1 bis SLG 4/SLA 4

Quittung

Byte

0	1	2	3
CHN	02	07	Status

CHN 01, 02, 03 oder 04 hex = SLG 1/SLA 1 bis SLG 4/SLA 4
Status MOBY-Status

A.7 MOBY F-Besonderheiten im read only-Betrieb (nur ASM 824/SLA 81 mit MDS F1xx)

Beim read only-Betrieb sind nur die Befehle RESET, STATUS und READ in einer bestimmten Form zugelassen.
Wie beim READ-/WRITE-Betrieb ist immer nur ein Befehl möglich, wobei RESET und STATUS immer abgesetzt werden können. Nach einem Hochlauf und RESET-Befehl werden alle Datenträger fortlaufend erfasst und im Telegrammpuffer (max. 50 Telegramme) abgelegt, wenn sie nicht zwischendurch mit einem READ-Befehl abgeholt werden.
Ist der Puffer voll, werden alle weiteren Datenträger ignoriert und mit dem letzten Telegramm wird eine Fehlermeldung übertragen.

Beim Erfassen eines neuen Datenträgers (mit anderen Daten) wird das folgende Telegramm in dem ASM zur Abholung bereitgestellt.

Befehl

READ (Befehl zur Sendetriggerung des ASM 824)

Byte

0	1	2	3	4	5	6
CHN	05	50	00	00	00	05

CHN 01, 02, 03 oder 04 hex = SLG 1/SLA 1 bis SLG 4/SLA 4

Quittung

Byte

0	1	2	3	4	5	6	7	8	9	10	11
CHN	0A	50	Status	HR/ANW	Zähler	05	Identnummer				

HR/ANW

Bit

7	6	5	4	3	2	1	0
0	0	0	0	0	0	HR	ANW

CHN	01, 02, 03 oder 04 hex = SLG 1/SLA 1 bis SLG 4/SLA 4
Status	MOBY-Status
HR	Historical Read, d.h., die Daten konnten gerade noch generiert werden, z. B. bei hoher Datenträgergeschwindigkeit.
ANW	0 = MDS nicht anwesend 1 = MDS anwesend
Zähler	Wird bei jedem neuen Telegramm fortlaufend um 1 hochgezählt (00 bis FF hex)
Identnummer	Identnummer des MDS F1xx (5 Byte)

Der nachfolgend aufgeführte Befehl READ ist kein Befehl wie unter A.6.4 beschrieben. Der Befehl dient zur Datenflusssteuerung. Die Daten werden vom ASM automatisch erfasst und müssen mit diesem Befehl fortlaufend eingelesen werden. Mit dem Befehl kann jeweils ein vorhandenes read only-Telegramm abgeholt werden. Falls kein Telegramm vorhanden ist, wird vom ASM eines gesendet, sobald ein neues ansteht.

A.8 Mobile Datenspeicher

Speichertypen

Dem Anwender stehen mobile Datenspeicher mit unterschiedlichen Speichern zur Verfügung.

Tabelle A-3 Speichergrößen und -typen

System	Speichergröße normal	Speichergröße mit ECC	Speichertyp	MDS-Typ
MOBY I	62 Byte	42 Byte*	RAM	z. B. MDS 114
MOBY I	128 Byte	112 Byte*	EEPROM	z. B. MDS 213 E
MOBY I	2 KByte	1,7 Byte*	RAM	z. B. MDS 302
MOBY I	8 KByte	7 KByte*	FRAM	z. B. MDS 401
MOBY I	32 KByte	28 KByte*	RAM	z. B. MDS 506
MOBY E	752 Byte	–[1]	EEPROM	MDS E6xx
MOBY F	5 Byte	–[1]	Festcode	MDS F1xx
MOBY F	32 Byte	–[1]	EEPROM	MDS F2xx
MOBY F	256 Byte	–[1]	EEPROM	MDS F4xx

* *Nettokapazität bei ECC-Betrieb*

1 Der ECC-Betrieb ist für MOBY E/F nicht zugelassen.

Die folgende Tabelle zeigt den Adressraum der einzelnen MDS-Varianten.

Tabelle A-4 Adressraum der MDS

Adressierung	Hexadezimal 16 Bit		Festpunktzahl 16 Bit	
	normal	mit ECC	normal	mit ECC
MOBY I	**62 Byte Datenspeicher mit RAM**			
Anfangsadresse	0000	0000	+0	+0
Endadresse	003D	0029	+61	+41
MOBY I	**128 Byte Datenspeicher mit EEPROM**			
Anfangsadresse	0000	0000	+0	+0
Endadresse	007F	006F	+127	+111
MOBY I	**2 KByte Datenspeicher mit RAM**			
Anfangsadresse	0000	0000	+0	+0
Endadresse	07FC	06F1	+2044	+1777
MOBY I	**8 KByte Datenspeicher mit EEPROM**			
Anfangsadresse	0000	0000	+0	+0
Endadresse	1FFC	1BF1	+8188	+7153
MOBY I	**32 KByte Datenspeicher mit RAM**			
Anfangsadresse	0000	0000	+0	+0
Endadresse	7FFC	6FF1	+32764	+28657
MOBY E	**752 Byte Datenspeicher mit EEPROM**			
Anfangsadresse	0000	–[1]	+0	–[1]
Endadresse	02EF	–[1]	751	–[1]
Seriennummer bei MOBY E auslesen *				
Anfangsadresse	1FF0	–[1]	8176	–[1]
Länge	4	–[1]	4	–[1]
MOBY E	**5 Byte MDS F1xx (Festcode)**			
Anfangsadresse	0000	–[1]	+0	–[1]
Endadresse	0004	–[1]	+4	–[1]
MOBY F	**32 Byte MDS F2xx EEPROM**			
Anfangsadresse	0010	–[1]	+16	–[1]
Endadresse	001F	–[1]	+31	–[1]
ID-Nr. (kann nur komplett ausgelesen werden)				
Anfangsadresse	0000	–[1]	+0	–[1]
Länge	4	–[1]	+4	–[1]
MOBY F	**192 Byte MDS F4xx EEPROM**			
Anfangsadresse	0040	–[1]	+64	–[1]
Endadresse	00FF	–[1]	+255	–[1]
ID-Nr. (kann nur komplett ausgelesen werden)				
Anfangsadresse	0000	–[1]	+0	–[1]
Länge	4	–[1]	+4	–[1]

* *In der Datendarstellung im DATDB gilt: 1. Byte = MSB, 4. Byte = LSB*

1 Der ECC-Betrieb ist für MOBY E/F nicht zugelassen.

Die Adressierung der Datenspeicher erfolgt wie in der Tabelle angegeben.

A.9 Status und Fehlercodes (ASM 424, ASM 724 und ASM 824)

Tabelle A-5 Status und Fehlercodes von ASM 424/724/824

Fehlercode (hex)	Blinken der LED-Anzeige (je Kanal)	Beschreibung
00	00	kein Fehler
–	01	siehe Kodierung 0F
01	02	Anwesenheitsfehler. Befehl nur teilweise bearbeitet Bei ASM 824 (MDS F1xx) 2 Datenträger im Feld
02	02	Anwesenheitsfehler. Datenträger wurde nicht bearbeitet Timeout bei Befehlsabarbeitung (MDS an Feldgrenze)
03	03	Fehler in der Verbindung zum SLG/SLA
04	04	MDS RAM Fehler (nicht initialisiert)
05	05	Unbekannter Befehl
06	06	Feldstörung am SLG (Bei MOBY F SLA 81: INIT-Fehler)
07	07	Zu viele Sendefehler
08	08	CRC-Sendefehler
09	09	CRC-Fehler beim Quittungsempfang (nur bei Initialisierung)
0A	10	MDS verweigert Initialisierung
0B	11	Timeout bei Initialisierung
0C	12	Speicher des MDS kann nicht beschrieben werden
0D	13	Adressfehler auf MDS MDS an Feldgrenze
0E	14	ECC-Fehler
0F	01	Hochlaufmeldung
10	16	NEXT-Befehl nicht möglich oder nicht zugelassen
12	18	Interner Firmwarefehler
13	19	Watchdog
14	20	Firmwarefehler (Checksummenerror-Telegramm, Stacküberlauf, Programmcodeveränderung, Timeout der Kanalverbindungen, ...)
15	21	Parametrierfehler
16	22	Ungeeignete Verbindungskonfigurierung
17	23	Protokollfehler
18	–	Nur RESET-Befehl zulässig
19	25	Pufferüberlauf, alle Telegrammpuffer belegt. Vorherige(r) Befehl(e) aktiv
1A	–	PROFIBUS bzw. 3964-Fehler (Busverbindung war unterbrochen).
1E	30	Telegramm falsch aufgebaut
1F	–	Befehl(e) wurde(n) mit RESET abgebrochen
20 (binär xx1x xxxx)	32	**Keine Fehlermeldung!** Tritt nur auf, wenn mit eingeschaltetem ECC-Treiber gearbeitet wird. Es zeigt an, dass der Treiber einen 1 Bit-Fehler erkannt und korrigiert hat. Die Lese- bzw. Schreibdaten sind in Ordnung.

Tabelle A-5 Status und Fehlercodes von ASM 424/724/824

Fehlercode (hex)	Blinken der LED-Anzeige (je Kanal)	Beschreibung
40 (binär x1xx xxxx)	64	**Keine Fehlermeldung!** Dieses Bit ist normalerweise immer gesetzt. Es ist reserviert für die Zustandsanzeige einer 2. Batterie auf dem MDS.
80 (binär 1xxx xxxx)	128	**Keine Fehlermeldung!** Batteriespannung des MDS ist unter Schwellwert gefallen. Es wird empfohlen, den MDS umgehend zu wechseln. Bei MDS-Typen mit EEPROM ist dieses Statusbit immer gesetzt. Bei SINUMERIK erfolgt die Batteriemeldung ohne die Kennung “F” in der IDENTIFIKATION. Zur Erkennung einer schlechten Batterie kann an einer Stelle im Gesamtsystem das “fnr”-Feld ausgewertet werden.

B Programmierung des SLG U92 auf Basis Betriebssystem oder Treiber 3964R

Für wen ist dieser Anhang gedacht?

Dieser Anhang ist gedacht für Anwender, die ihre Applikation direkt auf Betriebssystem- oder Treiberebene 3964R aufsetzen und nicht die C-Bibliothek MOBY API verwenden.
Weitere Informationen zur Treiberebene 3964R finden Sie auch im Benutzerhandbuch „3964R-Protokoll unter Windows NT 4.0/95“ (als PDF-Datei auf der CD „Software MOBY“ enthalten) /07/.

Achtung

Für Applikationen, die nicht auf das C-Interface MOBY API und nicht direkt auf den Treiber 3964R der C-Bibliothek MOBY API aufsetzen, ist ein entsprechender Treiber 3964R für die Zielhardware (serielle Ankopplung) zu verwenden, der die Anforderungen für das SLG U92 erfüllt. Das Verhalten des Treibers 3964R im SLG U92 ist der MOBY U-Dokumentation /06/ zu entnehmen.

B.1 Allgemeines zur Kommunikation des SLG U92

Für die Kommunikation mit den mobilen Datenträgern MDS U313, MDS U315, MDS U524, MDS U525 und MDS U589 und für die Steuerung des Systemverhaltens des SLG U92 stehen MDS- und Systemfunktionen zur Verfügung. Die Verwaltung der Daten auf den mobilen Datenspeichern MDS U313, MDS U315, MDS U524, MDS U525 und MDS U589 über byteweise Adressierung mit absoluten Adressen („Normaladressierung“).

Sie können zwischen drei Varianten der Systemsteuerung wählen:

1. MOBY I-aufrufkompatibel
   - Schreib-/Lesereichweite bis 1,5 m ⇒ Reichweitenbegrenzung auf 1,5 m fest eingestellt
   - Pulk/Multitag = 1 ⇒ Pulk auf 1 fest eingestellt
   - ohne BERO-Betrieb
     Das SLG U92 wertet nicht die digitalen Eingänge an der Serviceschnittstelle aus.
   - ohne SLG-Synchronisation
2. MOBY I-aufrufkompatibel mit erweiterten Befehlen
   - Schreib-/Lesereichweite von 0,5 m bis maximal 3,5 m in Stufen von 0,5 m einstellbar
   - Pulk/Multitag = 1 ⇒ Pulk auf 1 fest eingestellt
   - BERO-Betrieb möglich
   - SLG-Synchronisation möglich

3. MOBY U mit Multitag-Bearbeitung
   - Schreib-/Lesereichweite von 0,5 m bis maximal 3,5 m in Stufen von 0,5 m einstellbar
   - Pulk/Multitag ≤ 12
   - BERO-Betrieb möglich
   - SLG-Synchronisation möglich

Die Variante bestimmen Sie über das Systemtelegramm RESET (siehe Anhänge B.2.1.2.1, B.3.1.2.1 und B.4.1.2.1).

Die Funktionen werden zum SLG U92 als Telegramme mit dem Protokoll 3964R abgewickelt. Auf jedes Telegramm zum SLG U92 kommt eine Quittung mit oder ohne Nutzdaten vom SLG U92 zurück. Darüber hinaus können Meldungen als Telegramme vom SLG U92 azyklisch kommen.
Die Telegramme bestehen immer aus einem Telegrammkopf und haben je nach Funktion keine oder bis zu 251 Byte Nutzdaten. Ein Telegramm kann maximal 254 Byte lang sein.

Telegrammstruktur

	Telegrammkopf			**Nutzdaten (max. 251 Byte)**
Byte	0	1	2	3 bis max. 253
	AB [hex]	Befehl [hex]	Status [hex]	Nutzdaten [hex]

AB = Telegrammlänge in Bytes ohne das Byte AB

Befehl = Funktionskennung

Status = Statusfeld Status

Nutzdaten = Parameter, zu schreibende Daten auf MDS, ...
gelesene Daten vom MDS, Diagnosedaten, Statusdaten, ...

Für die Kommunikation mit dem SLG U92 ist der Treiber 3964R wie folgt zu konfigurieren:

- Data bits 8
- Stop bits 1
- Parity Odd
- Send buffer 255
- Receive buffer 255
- Baud rate 19200, 38400, 57600 oder 115200 Baud
- SLG U92 Slave mit automatischer Baudratenerkennung

B.2 MOBY I-aufrufkompatibel (Variante 1)

B.2.1 Telegramme an das SLG U92

MDS-Funktionen

- INIT MDS initialisieren
- WRITE Datenblock schreiben
- READ Datenblock lesen

Systemfunktionen

- RESET SLG rücksetzen
- SLG-STATUS SLG-Status/-Diagnose
- L-UEB Leitungsüberwachung

Mit dem Befehl RESET setzen Sie das SLG U92 in einen definierten Zustand zurück, und über die zu versorgenden Parameter bestimmen Sie das Systemverhalten des SLG U92.

Telegrammübersicht

	Telegrammkopf			**Nutzdaten (max. 251 Byte)**			
Byte	0	1	2	3 bis max. 253			
Telegramme (Funktion)	AB [hex]	Befehl [hex]	Status [hex]	Nutzdaten [hex]			
INIT	06	03	00	date	00	length	
WRITE	ABL	01	00	address	length	data	
READ	05	02	00	address	length		
RESET	05	00	00	standby	param	00	
SLG-STATUS	06	04	00	mode	00	00	00
L-UEB	02	FF	00				

AB = Telegrammlänge in Bytes ohne das Byte AB
ABL = variable Telegrammlänge in Bytes ohne das Byte AB, in Abhängigkeit des Parameters length $\Rightarrow$ 5 + length

Achtung

Die Daten sind in der Telegrammübersicht und in den nachfolgenden einzelnen Telegrammdarstellungen im hexadezimalen Format (hex) dargestellt.

B.2.1.1 MDS-Funktionen

Mit den MDS-Funktionen INIT, WRITE und READ schreiben oder lesen Sie Daten auf den oder vom MDS.

B.2.1.1.1 Funktion INIT

Mit der Funktion INIT initialisieren Sie mit einem Bitmuster den MDS, der sich im Antennenfeld des SLG U92 befindet. Es ist ein „ungezielter“ Initialisierungsaufruf, da der MDS nicht anhand der ID-Nummer identifiziert wird.

Byte	0	1	2	3	4	5 6
Parameter	06	03	00	date	00	length

date	binärer Wert	Bitmuster 00 hex bis FF hex, mit dem der Datenträger initialisiert (beschrieben) werden soll.		
length	binärer Wert	32768	=	Länge der Datenspeicher MDS U524, MDS U525 und MDS U589 in Bytes
		2048	=	Länge des Datenspeichers MDS U313 und MDS U315 in Bytes.

Der Befehl INIT darf nur an das SLG U92 abgesetzt werden, wenn noch kein Befehl beim SLG U92 ansteht. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.
Wenn sich mehr als ein MDS in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.
Wenn sich kein MDS in der Zone 1 befindet, so bleibt der Befehl anstehen, bis ein MDS in die Zone 1 oder der Befehl RESET kommt.

B.2.1.1.2 Funktion WRITE

Mit der Funktion WRITE schreiben Sie Daten auf den MDS, der sich im Antennenfeld des SLG U92 befindet. Es ist ein „ungezielter“ Schreibaufruf, da der MDS nicht anhand der ID-Nummer identifiziert wird.

Byte	0	1	2	3	4	5	6	bis maximal	253
Parameter	ABL	01	00	address		length		data	

ABL	binärer Wert	1 bis 253	=	Telegrammlänge in Bytes ohne das Byte AB.
address	binärer Wert	0 bis maximale Länge der Nutzdaten minus 1. Anfangsadresse auf dem MDS für die zu schreibenden Daten. Die Adresse plus Datenlänge muss kleiner sein als die Endadresse.		
		- 16	=	FFF0 hex Anfangsadresse des OTP-Speichers
length	binärer Wert	1 bis 248	=	Länge der zu schreibenden Nutzdaten. Startadresse + Länge muss kleiner dem Wert der Speicherlänge des MDS in Bytes minus 1 sein.
		16	=	Länge des OTP-Speichers
data	Binär-Information	auf den MDS zu schreibende Nutzdaten		

Der Befehl WRITE darf nur an das SLG U92 abgesetzt werden, wenn noch kein Befehl beim SLG U92 ansteht. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.
Wenn sich mehr als ein MDS in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.
Wenn sich kein MDS in der Zone 1 befindet, so bleibt der Befehl anstehen, bis ein MDS in die Zone 1 oder der Befehl RESET kommt.

Achtung

Die 128 Bit-Anwenderinformation im OTP-Speicher wird mit der Startadresse –16 (FFF0 hex) adressiert. Der OTP-Speicher kann nur einmal beschrieben werden. Bei diesem Schreibaufruf muss die vollständige Information mit 128 Bit Länge übergeben werden. Ein weiterer Schreibversuch wird mit Fehlermeldung abgewiesen.

B.2.1.1.3 Funktion READ

Mit der Funktion READ lesen Sie Daten vom MDS, der sich im Antennenfeld des SLG U92 befindet. Es ist ein „ungezielter“ Leseaufruf, da der MDS nicht anhand der ID-Nummer identifiziert wird.

Byte	0	1	2	3	4	5
Parameter	05	02	00	address		length

address	binärer Wert	0 bis maximale Länge der Nutzdaten minus 1. Anfangsadresse der zu lesenden Daten auf dem MDS. Die Adresse plus Datenlänge muss kleiner sein als die Endadresse.		
		- 16	=	FFF0 hex Anfangsadresse des OTP-Speichers
length	binärer Wert	1 bis 248	=	Länge der zu lesenden Nutzdaten. Startadresse + Länge muss kleiner dem Wert der Speicherlänge des MDS in Bytes minus 1 sein.
		16	=	Länge des OTP-Speichers

Der Befehl READ darf nur an das SLG U92 abgesetzt werden, wenn noch kein Befehl beim SLG U92 ansteht. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.
Wenn sich mehr als ein MDS in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.

Wenn sich kein MDS in der Zone 1 befindet, so bleibt der Befehl anstehen, bis ein MDS in die Zone 1 oder der Befehl RESET kommt.

Achtung

Die 128 Bit Anwenderinformation im OTP-Speicher wird mit der Startadresse –16 (FFF0 hex) adressiert. Die 128 Bit Anwenderinformation wird mit dem Befehl WRITE in den MDS geschrieben. Beim Leseaufruf muss die vollständige Information mit 128 Bit Länge angefordert werden.

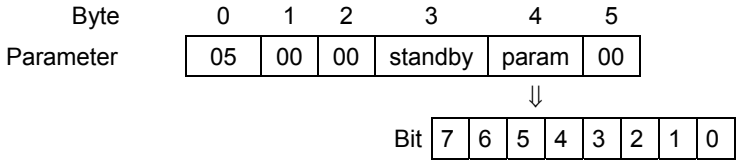
B.2.1.2 Systemfunktionen

B.2.1.2.1 Funktion RESET

Mit dem Befehl RESET setzen Sie das SLG U92 in einen definierten Zustand zurück, und über die zu versorgenden Parameter bestimmen Sie das Systemverhalten des SLG U92.

Standardeinstellungen:

- Schreib-/Lesereichweite bis 1,5 m
- Pulk/Multitag = 1
- ohne BERO-Betrieb, das SLG U92 wertet nicht die digitalen Eingänge an der Serviceschnittstelle aus



standby	binärer Wert	Standby-Zeit, für die der MDS nach einem ausgeführten MDS-Befehl in den Standby-Betrieb gehen soll. Das heißt, der MDS bleibt in dieser Zeit „wach“, um den nächsten Befehl, der innerhalb dieser Standby-Zeit kommen muss, ohne Verzögerung bearbeiten zu können. Der Wert gibt nicht direkt die Zeit, sondern den Faktor von 7 ms an. Z. B. der Wert 10 entspricht 10 x 7 ms = 70 ms.		
		0	=	kein Standby-Betrieb. Nach jeder Kommunikation mit dem MDS legt der MDS sich wieder „schlafen“.
		1 bis 200	=	7 ms bis 1400 ms
param	Bitmuster	Parameter		
		Bit 7 bis 6	=	0
		Bit 5	=	0 Betrieb ohne Anwesenheit
			=	1 Betrieb mit Anwesenheit (siehe Quittungstelegramm ANW-MELD)
		Bit 4	=	0 Reserve
		Bit 3 bis 0	=	5 hex Betriebsart MOBY U ⇒ MOBY I-Befehle ohne Erweiterungen

Der Befehl RESET darf zu jedem Zeitpunkt an das SLG U92 abgesetzt werden und wird sofort ausgeführt. Wenn ein anderer Befehl ansteht, so wird er abgebrochen. Nach der Ausführung des Befehls RESET ist die Antenne des SLG U92 eingeschaltet.

B.2.1.2.2 Funktion SLG-Status (SLG-Status/-Diagnose)

Mit dieser Funktion fragen Sie den Status des SLG U92 ab oder lesen Sie Diagnosedaten vom SLG U92 aus.

Byte	0	1	2	3	4	5	6
Parameter	06	04	00	mode	00	00	00

mode	binärer Wert	01 hex	=	SLG-Status
		02 hex	=	SLG-Diagnose I: letzte n Funktionsaufrufe anfordern
		03 hex	=	SLG-Diagnose II: letzte n Fehlermeldungen anfordern
		04 hex	=	SLG-Diagnose III: letzte n identifizierte MDS anfordern

Der Befehl SLG-STATUS darf zu jedem Zeitpunkt an das SLG U92 abgesetzt werden und wird sofort ausgeführt. Wenn ein Befehl wie WRITE, READ oder INIT beim SLG U92 ansteht, so bleibt er erhalten.

B.2.1.2.3 Funktion L-UEB

Mit dieser Funktion überwachen Sie die Verbindung zum SLG U92 auf logischer Ebene.

Byte	0	1	2
Parameter	02	FF	00

Der Befehl L-UEB darf zu jedem Zeitpunkt an das SLG U92 abgesetzt werden und wird sofort beantwortet. Wenn keine Rückmeldung kommt, so ist die Verbindung zum SLG U92 unterbrochen (gestört). Wenn ein Befehl wie WRITE, READ oder INIT beim SLG U92 ansteht, so bleibt er erhalten.

B.2.2 Quittungen/Meldungen vom SLG U92

Telegrammübersicht

	Telegrammkopf			**Nutzdaten (max. 251 Byte)**		
Byte	0	1	2	3 bis max. 253		
Quittung/ Meldung	AB [hex]	Befehl [hex]	Status [hex]	Nutzdaten [hex]		
INIT	02	03	00			
WRITE	02	01	00			
READ	ABL	02	00	address	length	data
RESET	05	00	00	FW	00	
SLG-STATUS (SLG-Status)	1B	04	00	S-Info	Statusinformation	
SLG-STATUS (Diagnose I)	ABL	04	00	S-Info	Diagnoseinformation	
SLG-STATUS (Diagnose II)	ABL	04	00	S-Info	Diagnoseinformation	
SLG-STATUS (Diagnose III)	ABL	04	00	S-Info	Diagnoseinformation	
L-UEB	02	FF	05			
Hochlauf	02	00	0F			
ANW-MELD	04	0F	00	00	ANW-S	

AB = Telegrammlänge in Bytes ohne das Byte AB
ABL = variable Telegrammlänge in Bytes ohne das Byte AB, in Abhängigkeit der variablen Nutzdatenlänge

B.2.2.1 Quittungen zu MDS-Funktionen

B.2.2.1.1 Quittung INIT

Byte	0	1	2
Parameter	02	03	status

status Bitmuster Status siehe Anhang B.6.

B.2.2.1.2 Quittung WRITE

Byte	0	1	2
Parameter	02	01	status

status Bitmuster Status siehe Anhang B.6.

B.2.2.1.3 Quittung READ

Quittung ohne Fehler (status gleich 00 hex)

Byte	0	1	2	3	4	5	6 bis maximal 253
Parameter	ABL	02	status	address		length	data

ABL	binärer Wert	1 bis 253 = Telegrammlänge in Bytes ohne das Byte AB.
status	Bitmuster	00 hex
address	binärer Wert	0 bis Wert: Speicherlänge in Bytes minus 1 -16 (FFF0 hex) nach Lesen des OTP-Speichers
length	binärer Wert	1 bis 248 = Länge der gelesenen Nutzdaten.
data	Binär-Information	vom MDS gelesene Nutzdaten

Quittung mit Fehler (status ungleich 00 hex)

Byte	0	1	2	3	4	5
Parameter	05	02	status	address		length

status	Bitmuster	Status siehe Anhang B.6.
address	binärer Wert	Im Funktionsaufruf vorgegebene Anfangsadresse auf dem MDS.
length	binärer Wert	Im Funktionsaufruf vorgegebene Länge der zu lesenden Daten auf dem MDS.

B.2.2.2 Quittungen zu Systemfunktionen

B.2.2.2.1 Quittung RESET

Byte	0	1	2	3	4	5
Parameter	05	00	status	FW-Stand		00

⇓

Byte	VersH	VersL

status	Bitmuster	Status siehe Anhang B.6.
VersH	binärer Wert	00 hex bis FF hex = Firmewarestand (High)
VersL	binärer Wert	00 hex bis FF hex = Firmewarestand (Low)
		z. B. 01 (High) und 0A (Low) = Version 1.10

B.2.2.2.2 Quittung SLG-STATUS (SLG-Status)

Quittung ohne Fehler (status gleich 00 hex)

Byte	0	1	2	3	4	5	6	7	8	9	10	11
Parameter	1B	04	status	S-Info	HW	HW-V		Url-V		FW	FW-V	

12	13	14	15	16	17	18	19	20	21	22
TR	TR-V		SS	Baud	00	00	00	dili	mtag	fcon

23	24	25	26	27
ftim	sema	ANT	standby	ANW

status	Bitmuster	00 hex
S-Info	binärer Wert	01 hex = Modus SLG-Status
HW	ASCII	HW-Variante
HW-V	binärer Wert	HW-Version 00 hex bis FF hex = Version (High-Byte) 00 hex bis FF hex = Version (Low-Byte)
Url-V	binärer Wert	Urlader-Version 00 hex bis FF hex = Version (High-Byte) 00 hex bis FF hex = Version (Low-Byte)
FW	ASCII-Format	FW-Variante
FW-V	binärer Wert	FW-Version 00 hex bis FF hex = Version (High-Byte) 00 hex bis FF hex = Version (Low-Byte)

TR	ASCII-Format	Treiber-Variante '1' = 3964R
TR-V	binärer Wert	Treiber-Version 00 hex bis FF hex = Version (High-Byte) 00 hex bis FF hex = Version (Low-Byte)
SS	binärer Wert	RS 232 / RS 422 01 hex = RS 422 02 hex = RS 232
Baud	binärer Wert	Baudrate 01 hex = 19,2 K Baud 02 hex = 38,4 K Baud 03 hex = 57,6 K Baud 05 hex = 115,2 K Baud
dili	binärer Wert	Reichenweitenbegrenzung (distance limiting) 05 hex = 0,5 m 0A hex = 1,0 m 0F hex = 1,5 m 14 hex = 2,0 m 19 hex = 2,5 m 1E hex = 3,0 m 23 hex = 3,5 m
mtag	binärer Wert	Anzahl der im Antennenfeld bearbeitbaren MDS (Multitag/Pulk) = 1
fcon	binärer Wert	BERO-Betriebsart (field ON control) 00 hex = Betriebsart 1: keine BEROs 01 hex = Betriebsart 2: ein oder zwei BEROs Die BEROs sind logisch ODER verknüpft. Während der Einschaltdauer des 1. und/oder 2. BEROs ist das Feld ein, sonst aus. 02 hex = Betriebsart 3: ein oder zwei BEROs Der 1. BERO schaltet das Feld ein und der 2. BERO schaltet das Feld aus. Wenn zwei BEROs vorhanden sind und eine BERO-Zeit parametriert ist, wird das Feld automatisch ausgeschaltet, wenn der 2. BERO nicht innerhalb dieser BERO-Zeit schaltet. Wenn der 2. BERO nicht vorhanden ist, muss eine BERO-Zeit parametriert sein. Nach dieser Zeit wird das Feld automatisch ausgeschaltet.
ftim	binärer Wert	BERO-Zeit (field ON time) 0 = keine BERO-Zeit (siehe BERO-Betrieb) 1 bis 255 = 1 bis 255 Sekunden
sema	binärer Wert	Semaphorensteuerung (Synchronisation mit SLG) 01 hex = ja 02 hex = nein

ANT	binärer Wert	Status Antenne	
		01 hex	= Antenne ein
		02 hex	= Antenne aus
standby	binärer Wert	Zeit nach einem ausgeführten MDS-Befehl für den Standby-Betrieb des MDS.	
		0	= kein Standby-Betrieb. Nach jeder Kommunikation mit dem MDS legt der MDS sich wieder „schlafen“. Erst nach der „Sleep time“ kann der MDS wieder gelesen oder beschrieben werden.
		1 bis 200	= 7 ms bis 1400 ms
ANW	binärer Wert	Anwesenheit (siehe RESET)	
		00 hex	= Betrieb ohne Anwesenheit
		01 hex	= Betrieb mit Anwesenheit (siehe Quittung ANW-MELD)

Quittung mit Fehler (status ungleich 00 hex)

Byte	0	1	2
Parameter	02	04	status

status Bitmuster Status siehe Anhang B.6.

B.2.2.2.3 Quittung SLG-STATUS (SLG-Diagnose I)

Quittung ohne Fehler (status gleich 00 hex)

Byte	0	1	2	3	4	5 bis 4 + 7	...	bis 4 + 7 * n
Parameter	ABL	04	status	S-Info	n	1. FKT (n = 1)	...	n. FKT (n = max.)

ABL	binärer Wert	Telegrammlänge in Bytes ohne das Byte AB 4 + 7 * n $0 \leq n \leq 33$ 4 bis maximal 236
status	Bitmuster	00 hex
S-Info	binärer Wert	02 hex = Modus SLG-Diagnose I
n	binärer Wert	Anzahl der zuletzt aufgerufenen Funktionen 0 bis 33
1. FKT	binärer Wert	1. Funktion: Funktionsdaten mit einer Länge von 7 Byte
„	„	„
n. FKT	binärer Wert	n. Funktion: Funktionsdaten mit einer Länge von 7 Byte

Quittung mit Fehler (status ungleich 00 hex)

Byte	0	1	2
Parameter	02	04	status

status Bitmuster Status siehe Anhang B.6.

B.2.2.2.4 Quittung SLG-STATUS (SLG-Diagnose II)

Quittung ohne Fehler (status gleich 00 hex)

Byte	0	1	2	3	4	4 + 1	....
Parameter	ABL	04	status	S-Info	n	1. FM (n = 1)	....

4 + n
n. FM (n = max.)

ABL	binärer Wert	Telegrammlänge in Bytes ohne das Byte AB 4 + n $0 \leq n \leq 233$ 4 bis maximal 238
status	Bitmuster	00 hex
S-Info	binärer Wert	03 hex = Modus SLG-Diagnose II
n	binärer Wert	Anzahl der zuletzt aufgetretenen Fehlermeldungen 0 bis 233
1. FM	binärer Wert	1. Fehlermeldung (-nummer) (1 Byte)
„	„	„
n. FM	binärer Wert	n. Fehlermeldung (-nummer) (1 Byte)

Quittung mit Fehler (status ungleich 00 hex)

Byte	0	1	2
Parameter	02	04	status

status	Bitmuster	Status siehe Anhang B.6.

B.2.2.2.5 Quittung SLG-STATUS (SLG-Diagnose III)

Quittung ohne Fehler (status gleich 00 hex)

Byte	0	1	2	3	4	5 bis 4 + 4	....
Parameter	ABL	04	status	S-Info	n	1. MDS-Nr.(n = 1)	....

bis 4 + 4 * n
n. MDS-Nr. (n = max.)

ABL	binärer Wert	Telegrammlänge in Bytes ohne das Byte AB 4 + 4 * n $0 \leq n \leq 24$ 4 bis maximal 100
status	Bitmuster	00 hex
S-Info	binärer Wert	04 hex = Modus SLG-Diagnose III
n	binärer Wert	Anzahl der letzten identifizierten MDS 0 bis 24
1. MDS-Nr.	binärer Wert	1. MDS-Nummer (4 Byte)
”	”	”
n. MDS-Nr.	binärer Wert	n. MDS-Nummer (4 Byte)

Quittung mit Fehler (status ungleich 00 hex)

Byte	0	1	2
Parameter	02	04	status

status	Bitmuster	Status siehe Anhang B.6.

B.2.2.2.6 Quittung L-UEB

Byte	0	1	2
Parameter	02	FF	05

B.2.2.3 Meldungen

B.2.2.3.1 Meldung Hochlauf

Byte	0	1	2
Parameter	02	00	0F

Das SLG U92 schickt ein Hochlauftelegramm nach dem Einschalten der Spannung am SLG U92.

B.2.2.3.2 Meldung ANW-MELD

Byte	0	1	2	3	4
Parameter	04	0F	status	00	ANW-S

status	Bitmuster	Status siehe Anhang B.6.
ANW-S	binärer Wert	Anwesenheitstatus = Anzahl der im Feld (Zone 1) befindlichen MDS. 0 bis 12

Wenn im Telegramm RESET das Bit „Betrieb mit Anwesenheit“ gesetzt wurde, dann schickt das SLG U92 nach jeder Anwesenheitsänderung im Feld (Zone 1) ein Telegramm mit der Anzahl der im Feld befindlichen MDS. Tritt gleichzeitig ein MDS aus dem Feld und ein anderer MDS in das Feld, so werden 2 Meldungen ANW-MELD verschickt. Treten gleichzeitig mehrere MDS ins Feld, so erzeugt jeder MDS eine Meldung ANW-MELD. Das Gleiche gilt für das Verlassen des Feldes.

Die Anwesenheitsmeldung kommt asynchron vom SLG U92.

B.3 MOBY I-aufrufkompatibel (Variante 2)

B.3.1 Telegramme an das SLG U92

MDS-Funktionen

- INIT MDS initialisieren
- WRITE Datenblock schreiben
- READ Datenblock lesen
- MDS-STATUS MDS-Status/-Diagnose

Systemfunktionen

- RESET SLG rücksetzen
- SLG-STATUS SLG-Status/-Diagnose
- SET-ANT Antenne ein-/ausschalten
- END Kommunikation mit MDS beenden
- REPEAT Letzten Befehl wiederholen
- L-UEB Leitungsüberwachung

Mit dem Befehl RESET setzen Sie das SLG U92 in einen definierten Zustand zurück, und über die zu versorgenden Parameter bestimmen Sie das Systemverhalten des SLG U92.

Telegrammübersicht

	Telegrammkopf			**Nutzdaten (max. 251 Byte)**						
Byte	0	1	2	3 bis max. 253						
Telegramm (Funktion)	AB [hex]	Befehl [hex]	Status [hex]	Nutzdaten [hex]						
INIT	06	03	00	date	00	length				
WRITE	ABL	01	00	address	length	data				
READ	05	02	00	address	length					
MDS-STATUS	05	0B	00	mode	cweek	year				
RESET	0A	00	00	standby	param	00	dili	01	fcon	ftim
SLG-STATUS	06	04	00	mode	00	00	00			
SET-ANT	03	0A	00	mode						
END	03	08	00	mode						
REPEAT	03	0D	00	mode						
L-UEB	02	FF	00							

AB	=	Telegrammlänge in Bytes ohne das Byte AB
ABL	=	variable Telegrammlänge in Bytes ohne das Byte AB, in Abhängigkeit des Parameters length ⇒ 5 + length

Achtung

Die Daten sind in der Telegrammübersicht und in den nachfolgenden einzelnen Telegrammdarstellungen im hexadezimalen Format (hex) dargestellt.

B.3.1.1 MDS-Funktionen

Mit den MDS-Funktionen INIT, WRITE und READ schreiben oder lesen Sie Daten auf oder vom MDS. Mit der MDS-Funktion MDS-STATUS fragen Sie die Status- und Diagnosedaten vom MDS ab.

B.3.1.1.1 Funktion INIT

Siehe Anhang B.2.1.1.1.

B.3.1.1.2 Funktion WRITE

Siehe Anhang B.2.1.1.2.

B.3.1.1.3 Funktion READ

Siehe Anhang B.2.1.1.3.

B.3.1.1.4 Funktion MDS-STATUS

Mit dieser Funktion erhalten Sie Status- und Diagnosedaten vom MDS, der sich im Antennenfeld des SLG U92 befindet. Es ist ein „ungezielter“ Leseaufruf, da der MDS nicht anhand der ID-Nummer identifiziert wird.

Byte	0	1	2	3	4	5
Parameter	05	0B	00	mode	cweek	year

mode	binärer Wert	00 hex	=	Status und Diagnosedaten eines MDS anfordern
cweek	binärer Wert	1 bis 53	=	aktuelle Kalenderwoche (Calendar Week)
year	binärer Wert	1 bis 99	=	aktuelle Jahreszahl (beginnend mit 1 für 2001)

Der Befehl MDS-STATUS darf nur dann an das SLG U92 abgesetzt werden, wenn kein Befehl beim SLG U92 ansteht.

Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.
Wenn sich kein MDS in der Zone 1 befindet, so erfolgt eine Fehlermeldung.
Wenn sich mehr als ein MDS in der Zone 1 befindet, so wird die Funktion mit Fehler abgebrochen.
Der Funktionsablauf ist von den Feldern cweek und year abhängig:

a) Ist der Wert in beiden Feldern innerhalb des Wertebereiches, dann wird in der Antwort die restliche Batterielebensdauer ausgegeben.

b) Ist einer der Werte außerhalb des angegebenen Wertebereiches, dann kann die restliche Batterielebensdauer nicht errechnet werden und die Funktion wird mit einem Fehler abgebrochen.

c) Sind beide Werte mit FF hex Tagen vorgegeben, so wird die restliche Batterielebensdauer nicht berechnet und in der Quittung wird als Batterielebensdauer FFFF hex angegeben.

B.3.1.2 Systemfunktionen

B.3.1.2.1 Funktion RESET

Mit dem Befehl RESET setzen Sie das SLG U92 in einen definierten Zustand zurück, und über die zu versorgenden Parameter bestimmen Sie das Systemverhalten des SLG U92.

Standardeinstellung:

- Pulk / Multitag = 1

Byte	0	1	2	3	4	5	6	7 8	9	10
Parameter	0A	00	00	standby	param	00	dili	00 01	fcon	ftim

⇓

Bit	7	6	5	4	3	2	1	0

standby binärer Wert Standby-Zeit, für die der MDS nach einem ausgeführten MDS-Befehl in den Standby-Betrieb gehen soll. Das heißt, der MDS bleibt in dieser Zeit „wach“, um den nächsten Befehl, der innerhalb dieser Standby-Zeit kommen muss, ohne Verzögerung bearbeiten zu können.
Der Wert gibt nicht direkt die Zeit, sondern den Faktor von 7 ms an. Z. B. der Wert 10 entspricht 10 x 7 ms = 70 ms.

0 = kein Standby-Betrieb. Nach jeder Kommunikation mit dem MDS legt der MDS sich wieder „schlafen“.

1 bis 200 = 7 ms bis 1400 ms

param	Bitmuster	Parameter			
		Bit 7 bis 6	=	0	
		Bit 5	=	0	Betrieb ohne Anwesenheit
			=	1	Betrieb mit Anwesenheit (siehe Quittung ANW-MELD)
		Bit 4	=	0	Reserve
		Bit 3 bis 0	=	5 hex	Betriebsart MOBY U ⇒ MOBY I-Befehle mit Erweiterungen

dili — binärer Wert

Reichenweitenbegrenzung (Zone 1):
Die Schreib-/Lesereichweite des SLG U92 (0,5 bis 3 m) kann in Stufen von 0,5 m begrenzt werden. Bei der maximalen Reichweite von 3 m muss als Begrenzung 3,5 m parametriert werden. Zusammen mit der Reichweitenbegrenzung kann die Sendeleistung reduziert werden. Gründe hierfür sind dem MOBY U-Handbuch für Projektierung, Montage und Service zu entnehmen.

normale Sendeleistung			reduzierte Sendeleistung		
05 hex	=	0,5 m	85 hex	=	0,5 m
0A hex	=	1,0 m	8A hex	=	1,0 m
0F hex	=	1,5 m	8F hex	=	1,5 m
14 hex	=	2,0 m	94 hex	=	2,0 m
19 hex	=	2,5 m	99 hex	=	2,5 m
1E hex	=	3,0 m	9E hex	=	3,0 m
23 hex	=	3,5 m	A3 hex	=	3,5 m

fcon — binärer Wert

BERO-Betriebsart

00 hex	=	Betriebsart 1: ohne BEROs oder SLG-Synchronisation
01 hex	=	Betriebsart 2: ein oder zwei BEROs Die BEROs sind logisch ODER-verknüpft. Während der Einschaltdauer des 1. und/oder 2. BEROs ist das Feld ein, sonst aus.
02 hex	=	Betriebsart 3: ein oder zwei BEROs Der 1. BERO schaltet das Feld ein und der 2. BERO schaltet das Feld aus. Wenn zwei BEROs vorhanden sind und eine BERO-Zeit parametriert ist, wird das Feld automatisch ausgeschaltet, wenn der 2. BERO nicht innerhalb dieser BERO-Zeit schaltet. Wenn der 2. BERO nicht vorhanden ist, muss eine BERO-Zeit parametriert sein. Nach dieser Zeit wird das Feld automatisch ausgeschaltet.
03 hex	=	Betriebsart 4: SLG-Synchronisation (siehe MOBY U-Handbuch für Projektierung, Montage und Service)

ftim — binärer Wert

0	=	BERO-Zeit = 0 (wenn BERO-Betriebsart = 0)
1 bis 255	=	BERO-Zeit = 1 bis 255 Sekunden

Der Befehl RESET darf zu jedem Zeitpunkt an das SLG U92 abgesetzt werden und wird sofort ausgeführt. Wenn ein anderer Befehl ansteht, so wird er abgebrochen. Nach der Ausführung des Befehls RESET ist die Antenne des SLG U92 eingeschaltet.

B.3.1.2.2 Funktion SLG-STATUS (SLG-Status/-Diagnose)

Siehe Anhang B.2.1.2.2.

B.3.1.2.3 Funktion SET-ANT

Mit dieser Funktion schalten Sie die Antenne des Schreib-/Lesegerätes (SLG U92) ein oder aus.

Byte	0	1	2	3
Parameter	03	0A	00	mode

mode binärer Wert 01 hex = Antenne einschalten
02 hex = Antenne ausschalten

Der Befehl SET-ANT darf nur an das SLG U92 abgesetzt werden, wenn noch kein Befehl beim SLG U92 ansteht.
Zum Zeitpunkt Antenne einschalten darf bereits ein MDS im Feld des SLG U92 vorhanden sein.
Wenn sich beim Abschalten ein MDS im Feld des SLG U92 befindet, so wird bei Betrieb mit Anwesenheit dieser als abwesend gemeldet.

B.3.1.2.4 Funktion END

Mit dieser Funktion schalten Sie für den zuletzt bearbeiteten und noch im Feld des SLG U92 befindlichen MDS die Standby-Zeit (im RESET-Telegramm parametriert) unwirksam, um den Stromverbrauch des MDS zu reduzieren.

Byte	0	1	2	3
Parameter	03	08	00	mode

mode	binärer Wert	00 hex	=	Die Bearbeitung mit dem MDS ist beendet. Der MDS wird das Feld des SLG U92 (Zone 1) verlassen. Mit diesem MDS soll keine Kommunikation mehr stattfinden. Die parametrierte Standby-Zeit wird unwirksam. Das SLG U92 trägt den MDS aus der Bearbeitungsliste aus und führt den MDS weiterhin in der Anwesenheitsliste bis der MDS die Zone 1 verlassen hat.
		01 hex	=	Bearbeitungpause mit dem MDS. Der MDS verlässt das Feld des SLG U92 (Zone 1) noch nicht. Es ist noch mindestens eine weitere Kommunikation mit dem MDS vorgesehen. Die parametrierte Standby-Zeit wird unwirksam. Das SLG U92 führt den MDS weiterhin in der Bearbeitungsliste und in der Anwesenheitsliste. Z. B. Befehl READ, Pause und anschließend Befehl WRITE

Der Befehl END darf nur nach dem Befehl WRITE, READ oder INIT an das SLG U92 abgesetzt werden, und es darf kein Befehl beim SLG U92 anstehen. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.

Dieser Befehl bezieht sich auf den zuletzt bearbeiteten MDS.
Wenn der MDS die Zone 1 bereits verlassen hat und der Modus 01 gewählt wurde, kommt eine Fehlermeldung. Ist dagegen ein weiterer MDS in die Zone 1 eingetreten und der Modus 01 gewählt, so kommt ebenso eine Fehlermeldung.

B.3.1.2.5 Funktion REPEAT

Mit dieser Funktion wird automatisch ein MDS-Befehl (MDS-Funktion) oder eine Befehlskette (MDS-Funktionen) wiederholt, sobald ein MDS in das Antennenfeld eintritt.

- MDS-Befehl: INIT, WRITE, READ und MDS-STATUS.

  Der Befehl END kann nicht automatisch wiederholt werden.
- Befehlskette: Verknüpfung aus den MDS-Befehlen INIT, WRITE, READ und MDS-STATUS und dem Befehl END.

  Der Befehl END darf nur am Ende der Befehlskette stehen.

Diese Funktion wiederholt den letzten übertragenen oder ausgeführten MDS-Befehl bzw. die letzte übertragene oder ausgeführte Befehlskette.

Byte	0	1	2	3
Parameter	03	0D	00	mode

mode	binärer Wert	00 hex	=	Wiederholen, bis dieser Befehl mit Modus gleich 1 kommt.
		01 hex	=	Wiederholung beenden. Ein begonnener Befehl wird zu Ende bearbeitet.

Der auszuführende MDS-Befehl oder die auszuführende Befehlskette muss korrekte Parameter enthalten oder bereits einmal fehlerfrei durchgeführt worden sein.

Wenn der MDS-Befehl oder die Befehlskette auf unterschiedliche MDS-Typen (2 kByte oder 32 kByte) angewendet werden soll, muss auf den zu adressierenden Datenbereich geachtet werden, sonst kann es zum Fehler 0D hex führen.
Der Modus Wiederholen wird auch nach dem Auftreten eines Fehlers beibehalten.

Achtung

Wenn die Funktion REPEAT nach einem der Befehle RESET, SLG-STATUS, SET-ANT oder END angestoßen wird, so wird sie mit einem Fehlerstatus abgewiesen.

Wenn die automatische Befehlswiederholung aktiviert ist und ein SLG-STATUS aufgerufen wird, so wird der SLG-STATUS asynchron ausgeführt. Die automatische Befehlswiederholung bleibt aktiv.

Wenn während der Befehlsausführung ein weiterer MDS in das Antennefeld eintritt, so wird die Befehlsausführung mit Fehler 1D hex abgebrochen. Das bedeutet bei einer Befehlskette, dass vom Fehlerzeitpunkt an jedes Telegramm mit Fehlerstatus quittiert wird. Wenn sich nur noch ein MDS im Feld befindet, so wird auf diesen MDS der Befehl oder die Befehlskette ausgeführt.

Wenn auf einen MDS, der sich im Antennenfeld befindet, der MDS-Befehl oder die Befehlskette ohne den Befehl END ausgeführt wurde und anschließend ein weiterer MDS in das Feld eintritt, so wird der auszuführende MDS-Befehl oder die Befehlskette mit Fehler 1D hex abgebrochen. Das bedeutet bei einer Befehlskette je Telegramm eine Quittung mit Fehlerstatus.

Vorsicht

Es wird nicht überprüft, ob in einem Schreibbefehl der OTP-Speicher adressiert wurde. Bei der automatischen Wiederholung würde jeder MDS den gleichen OTP-Speicherinhalt bekommen.

B.3.1.2.6 Funktion L-UEB

Siehe Anhang B.2.1.2.3.

B.3.2 Quittungen/Meldungen vom SLG U92

Telegrammübersicht

	Telegrammkopf			Nutzdaten (max. 251 Byte)			
Byte	0	1	2	3 bis max. 253			
Quittung/ Meldung	AB [hex]	Befehl [hex]	Status [hex]	Nutzdaten [hex]			
INIT	02	03	00				
WRITE	02	01	00				
READ	ABL	02	00	address	length	data	
MDS-STATUS	12	0B	00	MDS-Nr.	MDS-Typ	Σ Teilrahmenzugr.	
				Σ search modezugr.	Σ MCOD	Rest. B.	ST
RESET	05	00	00	FW	00		
SLG-STATUS (SLG-Status)	1B	04	00	S-Info	Statusinformation		
SLG-STATUS (Diagnose I)	ABL	04	00	S-Info	Diagnoseinformation		
SLG-STATUS (Diagnose II)	ABL	04	00	S-Info	Diagnoseinformation		
SLG-STATUS (Diagnose III)	ABL	04	00	S-Info	Diagnoseinformation		
SET-ANT	02	0A	00				
END	02	08	00				
REPEAT	02	0D	00				
L-UEB	02	FF	05				
Hochlauf	02	00	0F				
ANW-MELD	04	0F	00	00	ANW-S		

AB = Telegrammlänge in Bytes ohne das Byte AB
ABL = variable Telegrammlänge in Bytes ohne das Byte AB, in Abhängigkeit der variablen Nutzdatenlänge

B.3.2.1 Quittungen zu MDS-Funktionen

B.3.2.1.1 Quittung INIT

Siehe Anhang B.2.2.1.1.

B.3.2.1.2 Quittung WRITE

Siehe Anhang B.2.2.1.2.

B.3.2.1.3 Quittung READ

Siehe Anhang B.2.2.1.3.

B.3.2.1.4 Quittung MDS-STATUS

Quittung ohne Fehler (status gleich 00 hex)

Byte	0	1	2	3 bis 6	7	8 bis 11	12	13
Parameter	12	0B	status	MDS-Nr.	MDS-Typ	Σ Teilrah-menzugr.	Σ search mode-zugr.	

14	15	16	17	18
MCOD		Rest. B.		ST

status	Bitmuster	00 hex	
MDS-Nr.	binärer Wert	Wert von 2^0 bis 2^{31}	
MDS-Typ	binärer Wert	84 hex	= MDS mit 2 KByte mit ECC
		86 hex	= MDS mit 32 KByte mit ECC
Σ Teilrah-menzugr.	binärer Wert	Summe der Teilrahmenzugriffe 32 Bit	
Σ search mode-zugr.	binärer Wert	Summe der search mode Zugriffe	
		16 Bit	= Oberen 16 Bit eines 32 Bit Wertes, der die Anzahl der Search Zustände vor der letzten Änderung der Sleep-time anzeigt
MCOD	binärer Wert	Datum der letzten Änderung der Sleep-time	
		16 Bit	= Byte14: Kalenderwoche = Byte15: Kalenderjahr (ohne Jahrhundertangabe)

		Wenn keine Änderung der Sleep-time stattgefunden hat, werden die Kalenderwoche 01 und das Kalenderjahr 01 ausgegeben.
Rest. B.	binärer Wert	16 Bit = Restliche Batterielebensdauer in Prozent
ST	binärer Wert	Sleep-time: Eingestellter Wert im MDS
		0 = 20 ms +/- 6,7 ms
		1 = 40 ms +/- 13,3 ms
		2 = 80 ms +/- 26,7 ms
		3 = 160 ms +/- 53,3 ms
		4 = 320 ms +/- 106,7 ms (Defaultwert)
		5 = 640 ms +/- 213,3 ms
		6 = 1280 ms +/- 426,7 ms
		7 = 2560 ms +/- 853,3 ms
		Angeführter Toleranzbereich bei jedem MDS statistisch gleich verteilt.

Quittung mit Fehler (status ungleich 00 hex)

Byte	0	1	2
Parameter	02	0B	status

status	Bitmuster	Status siehe Anhang B.6.

B.3.2.2 Quittungen zu Systemfunktionen

B.3.2.2.1 Quittung RESET

Siehe Anhang B.2.2.2.1.

B.3.2.2.2 Quittung SLG-STATUS (SLG-Status)

Quittung ohne Fehler (status gleich 00 hex)

Byte	0	1	2	3	4	5	6	7	8	9	10	11
Parameter	1B	04	status	S-Info	HW	HW-V		Url-V		FW	FW-V	

12	13	14	15	16	17	18	19	20	21	22
TR	TR-V		SS	Baud	00	00	00	dili	mtag	fcon

23	24	25	26	27
ftim	sema	ANT	standby	ANW

status	Bitmuster	00 hex	
S-Info	binärer Wert	01 hex	= Modus SLG-Status
HW	ASCII	HW-Variante	
HW-V	binärer Wert	HW-Version	
		00 hex bis FF hex	= Version (High-Byte)
		00 hex bis FF hex	= Version (Low-Byte)
Url-V	binärer Wert	Urlader-Version	
		00 hex bis FF hex	= Version (High-Byte)
		00 hex bis FF hex	= Version (Low-Byte)
FW	ASCII-Format	FW-Variante	
FW-V	binärer Wert	FW-Version	
		00 hex bis FF hex	= Version (High-Byte)
		00 hex bis FF hex	= Version (Low-Byte)
TR	ASCII-Format	Treiber-Variante	
		‘1’ = 3964R	
TR-V	binärer Wert	Treiber-Version	
		00 hex bis FF hex	= Version (High-Byte)
		00 hex bis FF hex	= Version (Low-Byte)
SS	binärer Wert	RS 232 / RS 422	
		01 hex	= RS 422
		02 hex	= RS 232
Baud	binärer Wert	Baudrate	
		01 hex	= 19,2 K Baud
		02 hex	= 38,4 K Baud
		03 hex	= 57,6 K Baud
		05 hex	= 115,2 K Baud

dili binärer Wert Reichenweitenbegrenzung (distance limiting)

normale Sendeleistung			reduzierte Sendeleistung		
05 hex	=	0,5 m	85 hex	=	0,5 m
0A hex	=	1,0 m	8A hex	=	1,0 m
0F hex	=	1,5 m	8F hex	=	1,5 m
14 hex	=	2,0 m	94 hex	=	2,0 m
19 hex	=	2,5 m	99 hex	=	2,5 m
1E hex	=	3,0 m	9E hex	=	3,0 m
23 hex	=	3,5 m	A3 hex	=	3,5 m

mtag binärer Wert Anzahl der im Antennenfeld bearbeitbaren MDS (Multitag/Pulk)
= 1

fcon binärer Wert BERO-Betriebsart (field ON control)

00 hex = Betriebsart 1: keine BEROs oder SLG-Synchronisation

01 hex = Betriebsart 2: ein oder zwei BEROs
Die BEROs sind logisch ODER verknüpft. Während der Einschaltdauer des 1. und/oder 2. BEROs ist das Feld ein, sonst aus.

02 hex = Betriebsart 3: ein oder zwei BEROs
Der 1. BERO schaltet das Feld ein und der 2. BERO schaltet das Feld aus.

03 hex = Betriebsart 4: SLG-Synchronisation (siehe MOBY U-Handbuch für Projektierung, Montage und Service)

Wenn zwei BEROs vorhanden sind und eine BERO-Zeit parametriert ist, wird das Feld automatisch ausgeschaltet, wenn der 2. BERO nicht innerhalb dieser BERO-Zeit schaltet.
Wenn der 2. BERO nicht vorhanden ist, muss eine BERO-Zeit parametriert sein. Nach dieser Zeit wird das Feld automatisch ausgeschaltet.

ftim binärer Wert BERO-Zeit (field ON time)

0 = keine BERO-Zeit (siehe BERO-Betrieb)

1 bis 255 = 1 bis 255 Sekunden

sema binärer Wert Semaphorensteuerung (Synchronisation mit SLG)

01 hex = ja

02 hex = nein

ANT binärer Wert Status Antenne

01 hex = Antenne ein

02 hex = Antenne aus

standby binärer Wert Zeit nach einem ausgeführten MDS-Befehl für den Standby-Betrieb des MDS

0 = kein Standby-Betrieb. Nach jeder Kommunikation mit dem MDS legt der MDS sich wieder „schlafen“. Erst nach der „Sleep time“ kann der MDS wieder gelesen oder beschrieben werden.

1 bis 200 = 7 ms bis 1400 ms

ANW	binärer Wert	Anwesenheit (siehe RESET)	
		00 hex	= Betrieb ohne Anwesenheit
		01 hex	= Betrieb mit Anwesenheit (siehe Quittung ANW-MELD)

Quittung mit Fehler (status ungleich 00 hex)

Byte	0	1	2
Parameter	02	04	status

status Bitmuster Status siehe Anhang B.6.

B.3.2.2.3 Quittung SLG-STATUS (SLG-Diagnose I)

Siehe Anhang B.2.2.2.3.

B.3.2.2.4 Quittung SLG-STATUS (SLG-Diagnose II)

Siehe Anhang B.2.2.2.4.

B.3.2.2.5 Quittung SLG-STATUS (SLG-Diagnose III)

Siehe Anhang B.2.2.2.5.

B.3.2.2.6 Quittung SET-ANT

Byte	0	1	2
Parameter	02	0A	status

status Bitmuster Status siehe Anhang B.6.

B.3.2.2.7 Quittung END

Byte	0	1	2
Parameter	02	08	status

status Bitmuster Status siehe Anhang B.6.

B.3.2.2.8 Quittung REPEAT

Byte	0	1	2
Parameter	02	0D	status

status Bitmuster Status siehe Anhang B.6.

B.3.2.2.9 Quittung L-UEB

Siehe Anhang B.2.2.2.6.

B.3.2.3 Meldungen

B.3.2.3.1 Meldung Hochlauf

Siehe Anhang B.2.2.3.1.

B.3.2.3.2 Meldung ANW-MELD

Siehe Anhang B.2.2.3.2.

B.4 MOBY U mit Multitag-Bearbeitung (Variante 3)

B.4.1 Telegramme an das SLG U92

MDS-Funktionen

- INIT MDS initialisieren
- WRITE Datenblock schreiben
- READ Datenblock lesen
- GET MDS erfassen
- COPY Daten von MDS 1 nach MDS 2 kopieren
- MDS-STATUS MDS-Status/-Diagnose

Systemfunktionen

- RESET SLG rücksetzen
- SLG-STATUS SLG-Status/-Diagnose
- SET-ANT Antenne ein-/ausschalten
- END Kommunikation mit MDS beenden
- REPEAT Letzten Befehl wiederholen
- L-UEB Leitungsüberwachung

Mit dem Befehl RESET setzen Sie das SLG U92 in einen definierten Zustand zurück, und über die zu versorgenden Parameter bestimmen Sie das Systemverhalten des SLG U92.

Telegrammübersicht

	Telegrammkopf			Nutzdaten (max. 251 Byte)						
Byte	0	1	2	3 bis max. 253						
Telegramm (Funktion)	AB [hex]	Befehl [hex]	Status [hex]	Nutzdaten [hex]						
INIT	0A	03	00	MDS-Nr.	date	00	length			
WRITE	ABL	01	00	MDS-Nr.	address	length	data			
READ	09	02	00	MDS-Nr.	address	length				
GET	06	0C	00	mode	address	length				
COPY	10	07	00	MDS-Nr. 1	address1	length	MDS-Nr. 2	address2		
MDS-STATUS	09	0B	00	MDS-Nr.	mode	cweek	year			
RESET	0A	00	00	standby	param	00	dili	mtag	fcon	ftim
SLG-STATUS	06	04	00	mode	00	00	00			
SET-ANT	03	0A	00	mode						
END	07	08	00	MDS-Nr.	mode					
REPEAT	03	0D	00	mode						
L-UEB	02	FF	00							

AB = Telegrammlänge in Bytes ohne das Byte AB
ABL = variable Telegrammlänge in Bytes ohne das Byte AB, in Abhängigkeit des Parameters length ⇒ 5 + length

Achtung

Die Daten sind in der Telegrammübersicht und in den nachfolgenden einzelnen Telegrammdarstellungen im hexadezimalen Format (hex) dargestellt.

B.4.1.1 MDS-Funktionen

- Mit den MDS-Funktionen INIT, WRITE und READ schreiben oder lesen Sie Daten auf den oder vom MDS.
- Mit der Funktion GET fragen Sie ab, welche MDS sich im Feld befinden, gleichzeitig können Sie Daten von diesen MDS lesen.
- Mit der Funktion COPY kopieren Sie Daten: einen Datenbereich oder den kompletten Datenträgerinhalt von einem auf einen anderen MDS.
- Mit der Funktion MDS-STATUS fragen Sie die Status- und Diagnosedaten von einem MDS ab.

B.4.1.1.1 Funktion INIT

Mit der Funktion INIT initialisieren Sie „ungezielt“ oder „gezielt“ einen MDS mit einem Bitmuster, der sich im Antennenfeld des SLG U92 befindet.

- Es ist ein „ungezielter“ Initialisierungsaufruf, wenn der Aufruf ohne Identnummer des MDS abgesetzt wird. Es darf sich nur ein MDS im Antennenfeld befinden.
- Es ist ein „gezielter“ Initialisierungsaufruf, wenn der Aufruf mit Identnummer des MDS abgesetzt wird. Es dürfen sich mehrere MDS im Antennenfeld befinden. Die Identnummer des MDS können Sie über die Funktion GET ermitteln.

Byte	0	1	2	3 bis 6	7	8	9 10
Parameter	0A	03	00	MDS-Nr.	date	00	length

MDS-Nr.	binärer Wert	0	=	Wenn sich ein MDS im Antennenfeld befindet und ein „ungezielter“ Initialisierungsaufruf erfolgen soll.
		Wert 2^0 bis 2^{32} -1	=	MDS-Nr. von dem MDS, der initialisiert werden soll.
date	binärer Wert	Bitmuster 00 hex bis FF hex, mit dem der Datenträger initialisiert (beschrieben) werden soll.		
length	binärer Wert	32768	=	Länge der Datenspeicher MDS U524, MDS U525 und MDS U589 in Bytes
		2048	=	Länge des Datenspeichers MDS U313 und MDS U315 in Bytes.

Der Befehl INIT darf nur an das SLG U92 abgesetzt werden, wenn noch kein Befehl beim SLG U92 ansteht. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.
Wenn sich beim „ungezielten“ Aufruf mehr als ein MDS in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.
Wenn sich beim „ungezielten“ Aufruf kein MDS in der Zone 1 befindet, so bleibt der Befehl anstehen, bis ein MDS in die Zone 1 oder der Befehl RESET kommt.
Wenn sich beim „gezielten“ Aufruf der MDS mit der vorgegebenen MDS-Nr. nicht in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.

B.4.1.1.2 Funktion WRITE

Mit der Funktion WRITE schreiben Sie „ungezielt“ oder „gezielt“ Daten auf einen MDS, der sich im Antennenfeld des SLG U92 befindet.

- Es ist ein „ungezielter“ Schreibaufruf, wenn der Aufruf ohne Identnummer des MDS abgesetzt wird. Es darf sich nur ein MDS im Antennenfeld befinden.
- Es ist ein „gezielter“ Schreibaufruf, wenn der Aufruf mit Identnummer des MDS abgesetzt wird. Es dürfen sich mehrere MDS im Antennenfeld befinden. Die Identnummer des MDS können Sie über die Funktion GET ermitteln.

Byte	0	1	2	3 bis 6	7	8	9	10	bis maximal	253
Parameter	ABL	01	00	MDS-Nr.	address		length	data		

ABL	binärer Wert	1 bis 253	=	Telegrammlänge in Bytes ohne das Byte AB.
MDS-Nr.	binärer Wert	0	=	Wenn sich ein MDS im Antennenfeld befindet und ein „ungezielter“ Schreibaufruf erfolgen soll.
		Wert 2^0 bis 2^{32} -1	=	MDS-Nr. von dem MDS, auf den geschrieben werden soll.
address	binärer Wert	0 bis maximale Länge der Nutzdaten minus 1. Anfangsadresse auf dem MDS für die zu schreibenden Daten. Die Adresse plus Datenlänge muss kleiner sein als die Endadresse.		
		- 16	=	FFF0 hex Anfangsadresse des OTP-Speichers
length	binärer Wert	1 bis 244	=	Länge der zu schreibenden Nutzdaten.
		16	=	Länge des OTP-Speichers
data	Binär-Information	auf den MDS zu schreibende Nutzdaten		

Der Befehl WRITE darf nur an das SLG U92 abgesetzt werden, wenn noch kein Befehl beim SLG U92 ansteht. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.
Wenn sich beim „ungezielten“ Aufruf kein MDS in der Zone 1 befindet, so bleibt der Befehl anstehen, bis ein MDS in die Zone 1 oder der Befehl RESET kommt.
Wenn sich beim „ungezielten“ Aufruf mehr als ein MDS in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.
Wenn sich beim „gezielten“ Aufruf der MDS mit der vorgegebenen MDS-Nr. nicht in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.

Achtung

Die 128 Bit-Anwenderinformation im OTP-Speicher wird mit der Startadresse –16 (FFF0 hex) adressiert. Der OTP-Speicher kann nur einmal beschrieben werden. Bei diesem Schreibaufruf muss die vollständige Information mit 128 Bit Länge übergeben werden. Ein weiterer Schreibversuch wird mit Fehlermeldung abgewiesen.

B.4.1.1.3 Funktion READ

Mit der Funktion READ lesen Sie „ungezielt“ oder „gezielt“ Daten von einem MDS, der sich im Antennenfeld des SLG U92 befindet.

- Es ist ein „ungezielter“ Leseaufruf, wenn der Aufruf ohne Identnummer des MDS abgesetzt wird. Es darf sich nur ein MDS im Antennenfeld befinden.
- Es ist ein „gezielter“ Leseaufruf, wenn der Aufruf mit Identnummer des MDS abgesetzt wird. Es dürfen sich mehrere MDS im Antennenfeld befinden. Die Identnummer des MDS können Sie über die Funktion GET ermitteln.

Byte	0	1	2	3 bis 6	7	8	9
Parameter	09	02	00	MDS-Nr.	address		length

MDS-Nr.	binärer Wert	0	=	Wenn sich ein MDS im Antennenfeld befindet und ein „ungezielter“ Leseaufruf erfolgen soll.
		Wert 2^0 bis 2^{32} -1	=	MDS-Nr. von dem MDS, der gelesen werden soll.
address	binärer Wert	0 bis maximale Länge der Nutzdaten minus 1. Anfangsadresse der zu lesenden Daten auf dem MDS. Die Adresse plus Datenlänge muss kleiner sein als die Endadresse.		
		- 16	=	FFF0 hex Anfangsadresse des OTP-Speichers
length	binärer Wert	1 bis 244	=	Länge der zu lesenden Nutzdaten.
		16	=	Länge des OTP-Speichers

Der Befehl READ darf nur an das SLG U92 abgesetzt werden, wenn noch kein Befehl beim SLG U92 ansteht. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.
Wenn sich beim „ungezielten“ Aufruf mehr als ein MDS in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.
Wenn sich beim „ungezielten“ Aufruf kein MDS in der Zone 1 befindet, so bleibt der Befehl anstehen, bis ein MDS in die Zone 1 oder der Befehl RESET kommt.
Wenn sich beim „gezielten“ Aufruf der MDS mit der vorgegebenen MDS-Nr. nicht in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.

Achtung

Die 128 Bit-Anwenderinformation im OTP-Speicher wird mit der Startadresse –16 (FFF0 hex) adressiert. Die 128 Bit-Anwenderinformation wird mit dem Befehl WRITE in den MDS geschrieben. Beim Leseaufruf muss die vollständige Information mit 128 Bit Länge angefordert werden.

B.4.1.1.4 Funktion GET

Mit der Funktion GET fragen Sie ab, welche MDS sich im Feld befinden. Sie können gleichzeitig mit der Abfrage Daten von diesen MDS lesen.

Byte	0	1	2	3	4	5	6
Parameter	06	0C	00	mode	address		length

mode	binärer Wert	0	=	keine Daten vom MDS lesen
		1	=	Daten vom MDS lesen
address	binärer Wert	0	=	wenn keine Daten vom MDS
		0 bis maximale Länge der Nutzdaten minus 1. Anfangsadresse der zu lesenden Daten auf dem MDS. Die Adresse plus Datenlänge muss kleiner sein als die Endadresse.		
		- 16	=	FFF0 hex Anfangsadresse des OTP-Speichers
length	binärer Wert	1 bis x	=	Länge der zu lesenden Nutzdaten. x = (242 – (4 * Pulkgröße)) / Pulkgröße
		16	=	Länge des OTP-Speichers

Der Befehl GET darf nur an das SLG U92 abgesetzt werden, wenn noch kein Befehl beim SLG U92 ansteht. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.

Wenn sich mehr MDS in der Zone 1 befinden, als beim Parameter Pulk im RESET-Telegramm zugelassen ist, dann wird die Anzahl der MDS ohne Identnummer in einer Fehlermeldung mitgeteilt. Die gemeldeten MDS können anschließend nicht bearbeitet werden.

Achtung

Die 128 Bit-Anwenderinformation im OTP-Speicher wird mit der Startadresse –16 (FFF0 hex) adressiert. Die 128 Bit-Anwenderinformation wird mit dem Befehl WRITE in den MDS geschrieben. Beim GET-Aufruf muss die vollständige Information mit 128 Bit Länge angefordert werden.

B.4.1.1.5 Funktion COPY

Mit der Funktion COPY kopieren Sie „gezielt“ Daten von einem auf einen anderen MDS:

- einen Datenbereich oder
- den kompletten Datenträgerinhalt

Mit dieser Funktion können Daten ohne Umweg über das Anwenderprogramm von einem MDS als Quelle auf einen anderen MDS als Ziel geschrieben, d. h., kopiert werden.

Byte	0	1	2	3 bis 6	7 8	9 10	11 bis 14	15 16
Parameter	10	0C	00	MDS-Nr. 1	address1	length	MDS-Nr. 2	address2

MDS-Nr. 1	binärer Wert	Wert 2^0 bis 2^{32} -1 = MDS-Nr. vom MDS 1, von dem kopiert werden soll.
address1	binärer Wert	0 bis maximale Länge der Nutzdaten minus 1. Anfangsadresse der zu kopierenden Daten von dem MDS 1. Die Adresse plus Datenlänge muss kleiner sein als die Endadresse.
length	binärer Wert	Anzahl der zu kopierenden Bytes: 1 bis maximale Länge der Datenspeicher.
MDS-Nr. 2	binärer Wert	Wert 2^0 bis 2^{32} -1 = MDS-Nr. vom MDS 2, auf den kopiert werden soll.
address2	binärer Wert	0 bis maximale Länge der Nutzdaten minus 1. Anfangsadresse auf dem MDS 2, ab der die Daten geschrieben werden sollen. Die Adresse plus Datenlänge muss kleiner sein als die Endadresse.

Der Befehl COPY darf nur an das SLG U92 abgesetzt werden, wenn noch kein Befehl beim SLG U92 ansteht. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.

Wenn sich die zwei angegebenen MDS nicht in der Zone 1 befinden, so wird der Befehl mit einem Fehler abgebrochen.

Achtung

Der OTP-Speicher kann mit dem COPY-Befehl nicht kopiert werden.

B.4.1.1.6 Funktion MDS-STATUS

Mit dieser Funktion fragen Sie „ungezielt“ oder „gezielt“ Status- und Diagnosedaten von einem MDS ab, der sich im Antennenfeld des SLG U92 befindet.

- Es ist ein „ungezielter“ Leseaufruf, wenn der Aufruf ohne Identnummer des MDS abgesetzt wird. Es darf sich nur ein MDS im Antennenfeld befinden.
- Es ist ein „gezielter“ Leseaufruf, wenn der Aufruf mit Identnummer des MDS abgesetzt wird. Es dürfen sich mehrere MDS im Antennenfeld befinden. Die Identnummer des MDS können Sie über die Funktion GET ermitteln.

Byte	0	1	2	3 bis 6	7	8	9
Parameter	09	0B	00	MDS-Nr.	mode	cweek	year

MDS-Nr.	binärer Wert	0	=	Wenn sich ein MDS im Antennenfeld befindet und eine „ungezielte“ Statusabfrage erfolgen soll.
		Wert 2^0 bis 2^{32} -1	=	MDS-Nr. von dem MDS, der abgefragt werden soll.
mode	binärer Wert	00 hex	=	Status und Diagnosedaten eines MDS anfordern
cweek	binärer Wert	1 bis 53	=	aktuelle Kalenderwoche (Calendar Week)
year	binärer Wert	1 bis 99	=	aktuelle Jahreszahl (beginnend mit 1 für 2001)

Der Befehl MDS-STATUS darf nur an das SLG U92 abgesetzt werden, wenn noch kein Befehl beim SLG U92 ansteht.

Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.
Wenn sich beim „ungezielten“ Aufruf kein MDS in der Zone 1 befindet, so erfolgt eine Fehlermeldung.
Wenn sich beim „ungezielten“ Aufruf mehr als ein MDS in der Zone 1 befindet, so wird der Befehl mit Fehler abgebrochen.
Wenn sich beim „gezielten“ Aufruf der MDS mit der vorgegebenen MDS-Nr. nicht in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.

Der Funktionsablauf ist von den Feldern cweek und year abhängig:

a) Ist der Wert in beiden Feldern innerhalb des Wertebereiches, dann wird in der Antwort die restliche Batterielebensdauer ausgegeben.

b) Ist einer der Werte außerhalb des angegebenen Wertebereiches, dann kann die restliche Batterielebensdauer nicht errechnet werden und die Funktion wird mit einem Fehler abgebrochen.

c) Sind beide Werte mit FF hex Tagen vorgegeben, so wird die restliche Batterielebensdauer nicht berechnet und in der Quittung wird als Batterielebensdauer FFFF hex angegeben.

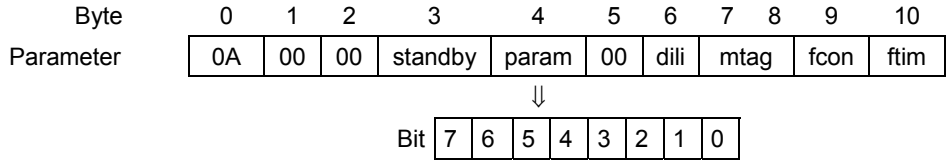
B.4.1.2 Systemfunktionen

B.4.1.2.1 Funktion RESET

Mit dem Befehl RESET setzen Sie das SLG U92 in einen definierten Zustand zurück, und über die zu versorgenden Parameter bestimmen Sie das Systemverhalten des SLG U92.

Standardeinstellung:

- Pulk / Multitag = 1



standby	binärer Wert	Standby-Zeit, für die der MDS nach einem ausgeführten MDS-Befehl in den Standby-Betrieb gehen soll. Das heißt, der MDS bleibt in dieser Zeit „wach“, um den nächsten Befehl, der innerhalb dieser Standby-Zeit kommen muss, ohne Verzögerung bearbeiten zu können. Der Wert gibt nicht direkt die Zeit, sondern den Faktor von 7 ms an. Z. B. der Wert 10 entspricht 10 x 7 ms = 70 ms.			
		0	=	kein Standby-Betrieb. Nach jeder Kommunikation mit dem MDS legt der MDS sich wieder „schlafen“.	
		1 bis 200	=	7 ms bis 1400 ms	
param	Bitmuster	Parameter			
		Bit 7 bis 6	=	0	
		Bit 5	=	0	Betrieb ohne Anwesenheit
			=	1	Betrieb mit Anwesenheit (siehe Quittung ANW-MELD)
		Bit 4	=	0	Reserve
		Bit 3 bis 0	=	6 hex	Betriebsart MOBY U ⇒ Multitag-Bearbeitung / Pulk
dili	binärer Wert	Reichenweitenbegrenzung (Zone 1): Die Schreib-/Lesereichweite des SLG U92 (0,5 bis 3 m) kann in Stufen von 0,5 m begrenzt werden. Bei der maximalen Reichweite von 3 m muss als Begrenzung 3,5 m parametriert werden. Zusammen mit der Reichweitenbegrenzung kann die Sendeleistung reduziert werden. Gründe hierfür sind dem MOBY U-Handbuch für Projektierung, Montage und Service zu entnehmen.			

		normale Sendeleistung		reduzierte Sendeleistung	
		05 hex	= 0,5 m	85 hex	= 0,5 m
		0A hex	= 1,0 m	8A hex	= 1,0 m
		0F hex	= 1,5 m	8F hex	= 1,5 m
		14 hex	= 2,0 m	94 hex	= 2,0 m
		19 hex	= 2,5 m	99 hex	= 2,5 m
		1E hex	= 3,0 m	9E hex	= 3,0 m
		23 hex	= 3,5 m	A3 hex	= 3,5 m

mtag	binärer Wert	Multitag / Pulk		
		1 bis 12	=	Anzahl der im Antennenfeld (Zone 1) bearbeitbaren MDS (Multitag / Pulk).
fcon	binärer Wert	BERO-Betriebsart		
		00 hex	=	Betriebsart 1: ohne BEROs oder SLG-Synchronisation
		01 hex	=	Betriebsart 2: ein oder zwei BEROs Die BEROs sind logisch ODER-verknüpft. Während der Einschaltdauer des 1. und/oder 2. BEROs ist das Feld ein, sonst aus.
		02 hex	=	Betriebsart 3: ein oder zwei BEROs Der 1. BERO schaltet das Feld ein und der 2. BERO schaltet das Feld aus. Wenn zwei BEROs vorhanden sind und eine BERO-Zeit parametriert ist, wird das Feld automatisch ausgeschaltet, wenn der 2. BERO nicht innerhalb dieser BERO-Zeit schaltet. Wenn der 2. BERO nicht vorhanden ist, muss eine BERO-Zeit parametriert sein. Nach dieser Zeit wird das Feld automatisch ausgeschaltet.
		03 hex	=	Betriebsart 4: SLG-Synchronisation (siehe MOBY U-Handbuch für Projektierung, Montage und Service)
ftim	binärer Wert	0	=	BERO-Zeit = 0 (wenn BERO-Betriebsart = 0)
		1 bis 255	=	BERO-Zeit = 1 bis 255 Sekunden

Der Befehl RESET darf zu jedem Zeitpunkt an das SLG U92 abgesetzt werden und wird sofort ausgeführt. Wenn ein anderer Befehl ansteht, so wird er abgebrochen. Nach der Ausführung des Befehls RESET ist die Antenne des SLG U92 eingeschaltet.

B.4.1.2.2 Funktion SLG-STATUS (SLG-Status/-Diagnose)

Siehe Anhang B.3.1.2.2

B.4.1.2.3 Funktion SET-ANT

Mit dieser Funktion schalten Sie die Antenne des Schreib-/Lesegerätes (SLG U92) ein oder aus.

Byte	0	1	2	3
Parameter	03	0A	00	mode

mode binärer Wert 01 hex = Antenne einschalten
02 hex = Antenne ausschalten

Der Befehl SET-ANT darf nur an das SLG U92 abgesetzt werden, wenn noch kein Befehl beim SLG U92 ansteht.
Zum Zeitpunkt Antenne einschalten darf bereits ein oder dürfen mehrere MDS im Feld des SLG U92 vorhanden sein.
Wenn sich beim Abschalten ein oder mehrere MDS im Feld des SLG U92 befinden, so wird bei Betrieb mit Anwesenheit jeder MDS einzeln als abwesend gemeldet.

B.4.1.2.4 Funktion END

Mit dieser Funktion schalten Sie die Standby-Zeit (im RESET-Telegramm parametriert) von einem MDS, der sich im Antennenfeld befindet, unwirksam, um den Stromverbrauch des MDS zu reduzieren.

Byte	0	1	2	3 bis 6	7
Parameter	07	08	00	MDS-Nr.	mode

MDS-Nr.	binärer Wert	0	=	Wenn sich ein MDS im Antennenfeld befindet und eine „ungezielter“ Aufruf erfolgen soll.
		Wert 2^0 bis 2^{32} -1	=	MDS-Nr. von dem MDS, bei dem die Standby-Zeit unwirksam geschaltet werden soll.
mode	binärer Wert	00 hex	=	Die Bearbeitung mit dem MDS ist beendet. Der MDS wird das Feld des SLG U92 (Zone 1) verlassen. Mit diesem MDS soll keine Kommunikation mehr stattfinden. Die parametrierte Standby-Zeit wird unwirksam. Das SLG U92 trägt den MDS aus der Bearbeitungsliste aus und führt den MDS weiterhin in der Anwesenheitsliste bis der MDS die Zone 1 verlassen hat.

01 hex	=	Bearbeitungpause mit dem MDS. Der MDS verlässt das Feld des SLG U92 (Zone 1) noch nicht. Es ist noch mindestens eine weitere Kommunikation mit dem MDS vorgesehen. Die parametrierte Standby-Zeit wird unwirksam. Das SLG U92 führt den MDS weiterhin in der Bearbeitungsliste und in der Anwesenheitsliste. Z. B. Befehl READ, Pause und anschließend Befehl WRITE

Der Befehl END darf nur nach dem Befehl WRITE, READ oder INIT an das SLG U92 abgesetzt werden, und es darf kein Befehl beim SLG U92 anstehen. Die Antenne muss eingeschaltet sein, sonst folgt eine Fehlermeldung.

Wenn sich beim „ungezielten“ Aufruf mehr als ein MDS in der Zone 1 befindet oder der zuletzt bearbeitete MDS die Zone 1 verlassen hat und der Modus 01 gewählt wurde, so wird der Befehl mit einem Fehler abgebrochen.

Wenn sich beim „gezielten“ Aufruf der MDS mit der vorgegebenen MDS-Nr. nicht in der Zone 1 befindet, so wird der Befehl mit einem Fehler abgebrochen.

B.4.1.2.5 Funktion REPEAT

Mit dieser Funktion wird automatisch ein MDS-Befehl (MDS-Funktion) oder eine Befehlskette (MDS-Funktionen) wiederholt, sobald ein MDS in das Antennenfeld eintritt.

- MDS-Befehl: INIT, WRITE, READ und MDS-STATUS.

  Die Befehle GET, COPY und END können nicht automatisch wiederholt werden.

- Befehlskette: Verknüpfung aus den MDS-Befehlen INIT, WRITE, READ und MDS-STATUS und dem Befehl END.

  Der Befehl END darf nur am Ende der Befehlskette stehen.

Diese Funktion wiederholt den letzten übertragenen oder ausgeführten MDS-Befehl bzw. die letzte übertragene oder ausgeführte Befehlskette.

Bei dem (den) auszuführenden MDS-Befehl(en) muss die MDS-Nr. mit Null vorbesetzt sein. Es darf keine MDS-Nr. eingetragen werden.
Die Funktion REPEAT ist möglich, wenn

- Pulk = 1 parametriert ist und sich ein MDS im Antennenfeld befindet oder
- Pulk > 1 parametriert ist und sich nur ein MDS im Antennenfeld befindet.

Byte	0	1	2	3
Parameter	03	0D	00	mode

mode binärer Wert 00 hex = Wiederholen, bis dieser Befehl mit Modus gleich 1 kommt.

01 hex = Wiederholung beenden. Ein begonnener Befehl wird zu Ende bearbeitet.

Der auszuführende MDS-Befehl oder die auszuführende Befehlskette muss korrekte Parameter enthalten oder bereits einmal fehlerfrei durchgeführt worden sein.

Wenn der MDS-Befehl oder die Befehlskette auf unterschiedliche MDS-Typen (2 kByte oder 32 kByte) angewendet werden soll, muss auf den zu adressierenden Datenbereich geachtet werden, sonst kann es zum Fehler 0D hex führen.
Der Modus Wiederholen wird auch nach dem Auftreten eines Fehlers beibehalten.

Achtung

Wenn die Funktion REPEAT nach einem der Befehle RESET, SLG-STATUS, SET-ANT, GET, COPY oder END angestoßen wird, so wird sie mit einem Fehlerstatus abgewiesen.

Wenn die automatische Befehlswiederholung aktiviert ist und ein SLG-STATUS aufgerufen wird, so wird der SLG-STATUS asynchron ausgeführt. Die automatische Befehlswiederholung bleibt aktiv.

Wenn während der Befehlsausführung ein weiterer MDS in das Antennefeld eintritt, so wird die Befehlsausführung mit Fehler 1D hex abgebrochen. Das bedeutet bei einer Befehlskette, dass vom Fehlerzeitpunkt an jedes Telegramm mit Fehlerstatus quittiert wird. Wenn sich nur noch ein MDS im Feld befindet, so wird auf diesen MDS der Befehl oder die Befehlskette ausgeführt.

Wenn auf einen MDS, der sich im Antennenfeld befindet, der MDS-Befehl oder die Befehlskette ohne den Befehl END ausgeführt wurde und anschließend ein weiterer MDS in das Feld eintritt, so wird der auszuführende MDS-Befehl oder die Befehlskette mit Fehler 1D hex abgebrochen. Das bedeutet bei einer Befehlskette je Telegramm eine Quittung mit Fehlerstatus.

Vorsicht

Es wird nicht überprüft, ob in einem Schreibbefehl der OTP-Speicher adressiert wurde. Bei der automatischen Wiederholung würde jeder MDS den gleichen OTP-Speicherinhalt bekommen.

B.4.1.2.6 Funktion L-UEB

Siehe Anhang B.2.1.2.3

B.4.2 Quittungen/Meldungen vom SLG U92

Telegrammübersicht

	Telegrammkopf			Nutzdaten (max. 251 Byte)			
Byte	0	1	2	3 bis max. 253			
Quittung/ Meldung	AB [hex]	Befehl [hex]	Status [hex]	Nutzdaten [hex]			
INIT	06	03	00	MDS-Nr.			
WRITE	06	01	00	MDS-Nr.			
READ	ABL	02	00	MDS-Nr.	address	length	data
GET	ABL	0C	00	Anz. MDS	1. MDS-Nr.	…	n. MDS-Nr.
				address	length	data MDS 1	
				…	data MDS n		
COPY	0A	07	00	MDS-Nr. 1	MDS-Nr. 2		
MDS-STATUS	12	0B	00	MDS-Nr.	MDS-Typ	Σ Teilrahmenzugr.	
				Σ search modezugr.	Σ MCOD	Rest. B.	ST
RESET	05	00	00	FW	00		
SLG-STATUS (SLG-Status)	1B	04	00	S-Info	Statusinformation		
SLG-STATUS (Diagnose I)	ABL	04	00	S-Info	Diagnoseinformation		
SLG-STATUS (Diagnose II)	ABL	04	00	S-Info	Diagnoseinformation		
SLG-STATUS (Diagnose III)	ABL	04	00	S-Info	Diagnoseinformation		
SET-ANT	02	0A	00				
END	06	08	00	MDS-Nr.			
REPEAT	02	0D	00				
L-UEB	02	FF	05				
Hochlauf	02	00	0F				
ANW-MELD	04	0F	00	00	ANW-S		

AB = Telegrammlänge in Bytes ohne das Byte AB
ABL = variable Telegrammlänge in Bytes ohne das Byte AB, in Abhängigkeit der variablen Nutzdatenlänge

B.4.2.1 Quittungen zu MDS-Funktionen

B.4.2.1.1 Quittung INIT

Quittung ohne Fehler (status gleich 00 hex)

Byte	0	1	2	3 bis 6
Parameter	06	03	status	MDS-Nr.

status	Bitmuster	00 hex		
MDS-Nr.	binärer Wert	Wert 2^0 bis 2^{32} -1	=	MDS-Nr. des initialisierten MDS

Quittung mit Fehler (status ungleich 00 hex)

Byte	0	1	2	3 bis 6
Parameter	06	03	status	MDS-Nr.

status	Bitmuster	Status siehe Anhang B.6.		
MDS-Nr.	binärer Wert	Wert 2^0 bis 2^{32} -1	=	MDS-Nr. des nicht initialisierten MDS („gezielter“ Initialisierungsaufruf mit Fehler)
		0	=	„ungezielter“ Initialisierungsaufruf mit Fehler

B.4.2.1.2 Quittung WRITE

Quittung ohne Fehler (status gleich 00 hex)

Byte	0	1	2	3 bis 6
Parameter	06	01	status	MDS-Nr.

status	Bitmuster	00 hex		
MDS-Nr.	binärer Wert	Wert 2^0 bis 2^{32} -1	=	MDS-Nr. des beschriebenen MDS

Quittung mit Fehler (status ungleich 00 hex)

Byte	0	1	2	3 bis 6
Parameter	06	01	status	MDS-Nr.

status	Bitmuster	Status siehe Anhang B.6.		
MDS-Nr.	binärer Wert	Wert 2^0 bis 2^{32} -1	=	MDS-Nr. des nicht beschriebenen MDS („gezielter“ Schreibaufruf mit Fehler)
		0	=	„ungezielter“ Schreibaufruf mit Fehler

B.4.2.1.3 Quittung READ

Quittung ohne Fehler (status gleich 00 hex)

Byte	0	1	2	3 bis 6	7 8	9	10 bis maximal 253
Parameter	ABL	02	status	MDS-Nr.	address	length	data

ABL	binärer Wert	1 bis 253	=	Telegrammlänge in Bytes ohne das Byte AB.
status	Bitmuster	00 hex		
MDS-Nr.	binärer Wert	Wert 2^0 bis 2^{32} -1	=	MDS-Nr. des gelesenen MDS
address	binärer Wert	0 bis Wert: Speicherlänge in Bytes minus 1		
		-16 (FFF0 hex) nach Lesen des OTP-Speichers		
length	binärer Wert	1 bis 244	=	Länge der gelesenen Nutzdaten.
data	Binär-Information	vom MDS gelesene Nutzdaten		

Quittung mit Fehler (status ungleich 00 hex)

Byte	0	1	2	3 bis 6
Parameter	06	02	status	MDS-Nr.

status	Bitmuster	Status siehe Anhang B.6.		
MDS-Nr.	binärer Wert	Wert 2^0 bis 2^{32} -1	=	MDS-Nr. des nicht gelesenen MDS („gezielter“ Leseaufruf mit Fehler)
		0	=	„ungezielter“ Leseaufruf mit Fehler

B.4.2.1.4 Quittung GET

Quittung ohne Fehler (status gleich 00 hex)

Je nach Funktionsaufruf und ob sich MDS im Antennenfeld (Zone 1) befinden, hat die Quittung folgende Strukturen:

Funktion GET mit mode = 0 (ohne MDS-Daten) und kein MDS im Feld

Byte	0	1	2	3
Parameter	03	0C	status	Anz. MDS

status	Bitmuster	00 hex		
Anz. MDS	binärer Wert	0	=	kein MDS im Feld

Funktion GET mit mode = 0 (ohne MDS-Daten) und mindestens 1 MDS im Antennenfeld (Zone 1)

Byte	0	1	2	3	4	bis	3 + 4 * n
Parameter	ABL	0C	status	Anz. MDS	1. MDS-Nr.	...	n. MDS-Nr.

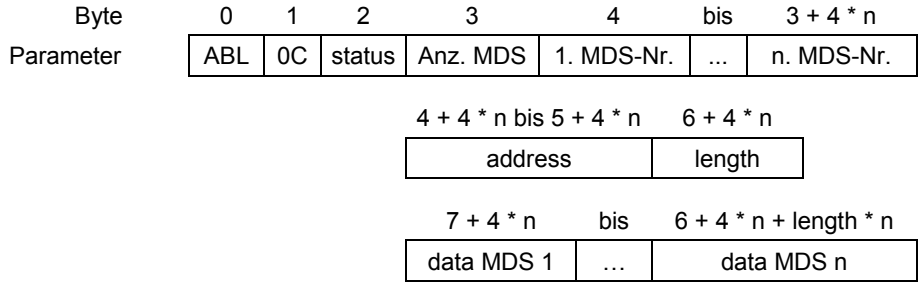
ABL	binärer Wert	Telegrammlänge in Bytes ohne das Byte AB.		
		7 bis 3 + 4 * n		1 < n ≤ 12; n = Anzahl MDS im Antennenfeld (Zone 1); je MDS-Nr. 4 Byte
status	Bitmuster	00 hex		
Anz. MDS	binärer Wert	0 bis 12	=	Anzahl der im Antennenfeld (Zone 1) vorhandenen MDS = n
1. MDS-Nr.	binärer Wert	Wert 2^0 bis 2^{32} -1	=	MDS-Nr. des 1. MDS
...				
n. MDS-Nr.	binärer Wert	Wert 2^0 bis 2^{32} -1	=	MDS-Nr. des n. MDS

Funktion GET mit mode = 1 (mit MDS-Daten) und kein MDS im Feld

Byte	0	1	2	3
Parameter	03	0C	status	Anz. MDS

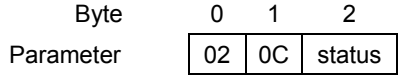
status	Bitmuster	00 hex		
Anz. MDS	binärer Wert	0	=	kein MDS im Feld

Funktion GET mit mode = 1 (mit MDS-Daten) und mindestens 1 MDS im Antennenfeld (Zone 1)



ABL	binärer Wert	Telegrammlänge in Bytes ohne das Byte AB.		
		14 bis 6 + 4 * n + length * n		1 < n ≤ 12; n = Anzahl MDS im Antennenfeld (Zone 1); je MDS-Nr. 4 Byte
status	Bitmuster	00 hex		
Anz. MDS	binärer Wert	0 bis 12	=	Anzahl der im Antennenfeld (Zone 1) vorhandenen MDS = n
1. MDS-Nr.	binärer Wert	Wert 2^0 bis 2^{32} -1	=	MDS-Nr. des 1. MDS
...				
n. MDS-Nr.	binärer Wert	Wert 2^0 bis 2^{32} -1	=	MDS-Nr. des n. MDS
address	binärer Wert	siehe Funktion GET, Anhang B.4.1.1.4		
length	binärer Wert	siehe Funktion GET, Anhang B.4.1.1.4		
Data MDS 1	Binär-Information	vom MDS 1 gelesene Nutzdaten		
…				
Data MDS n	Binär-Information	vom MDS n gelesene Nutzdaten		

Quittung mit Fehler (status ungleich 00 hex)



status	Bitmuster	Status siehe Anhang B.6.

B.4.2.1.5 Quittung COPY

Quittung ohne Fehler (status gleich 00 hex)

Byte	0	1	2	3 bis 6	7 bis 10
Parameter	0A	07	status	MDS-Nr. 1	MDS-Nr. 2

status	Bitmuster	00 hex		
MDS-Nr. 1	binärer Wert	Wert 2^0 bis 2^{32} -1	=	MDS-Nr. des gelesenen MDS (Quelle)
MDS-Nr. 2	binärer Wert	Wert 2^0 bis 2^{32} -1	=	MDS-Nr. des beschriebenen MDS (Ziel)

Quittung mit Fehler (status ungleich 00 hex)

Byte	0	1	2	3 bis 6	7 bis 10
Parameter	0A	07	status	MDS-Nr. 1	MDS-Nr. 2

status	Bitmuster	Status siehe Anhang B.6.		
MDS-Nr. 1	binärer Wert	Wert 2^0 bis 2^{32} -1	=	MDS-Nr. des nicht gelesenen MDS (Quelle)
		0	=	MDS-Nr. 1 nicht versorgt
MDS-Nr. 2	binärer Wert	Wert 2^0 bis 2^{32} -1	=	MDS-Nr. des nicht beschriebenen MDS (Ziel)
		0	=	MDS-Nr. 2 nicht versorgt

B.4.2.1.6 Quittung MDS-STATUS

Siehe Anhang B.3.2.1.4.

B.4.2.2 Quittungen zu Systemfunktionen

B.4.2.2.1 Quittung RESET

Siehe Anhang B.2.2.2.1.

B.4.2.2.2 Quittung SLG-STATUS (SLG-Status)

Siehe Anhang B.3.2.2.2.

B.4.2.2.3 Quittung SLG-STATUS (SLG-Diagnose I)

Siehe Anhang B.2.2.2.3.

B.4.2.2.4 Quittung SLG-STATUS (SLG-Diagnose II)

Siehe Anhang B.2.2.2.4.

B.4.2.2.5 Quittung SLG-STATUS (SLG-Diagnose III)

Siehe Anhang B.2.2.2.5.

B.4.2.2.6 Quittung SET-ANT

Siehe Anhang B.3.2.2.6.

B.4.2.2.7 Quittung END

Quittung ohne Fehler (status gleich 00 hex)

Byte	0	1	2	3 bis 6
Parameter	06	08	status	MDS-Nr.

status	Bitmuster	00 hex		
MDS-Nr.	binärer Wert	Wert 2^0 bis 2^{32} -1	=	MDS-Nr.

Quittung mit Fehler (status ungleich 00 hex)

Byte	0	1	2	3 bis 6
Parameter	06	08	status	MDS-Nr.

status	Bitmuster	Status siehe Anhang B.6.		
MDS-Nr.	binärer Wert	Wert 2^0 bis 2^{32} -1	=	MDS-Nr. („gezielter“ Aufruf mit Fehler)
		0	=	MDS-Nr. nicht versorgt („ungezielter“ Aufruf mit Fehler)

B.4.2.2.8 Quittung REPEAT

Siehe Anhang B.3.2.2.8.

B.4.2.2.9 Quittung L-UEB

Siehe Anhang B.2.2.2.6.

B.4.2.3 Meldungen

B.4.2.3.1 Meldung Hochlauf

Siehe Anhang B.2.2.3.1.

B.4.2.3.2 Meldung ANW-MELD

Siehe Anhang B.2.2.3.2.

B.5 Befehlskettung

Mit Hilfe der Befehlskettung können große und/oder verschiedene Adressbereiche auf dem MDS schneller bearbeitet werden.

Das SLG U92 hält im Normalfall nur einen MDS-Befehl im Speicher vor und arbeitet ihn ab. Das bedeutet, mit einem Befehl können maximal 248 Byte (ohne Multitag) bzw. 244 Byte (mit Multitag) geschrieben oder gelesen werden. Ein weiterer Befehl kann erst nach der Quittung auf den vorherigen Befehl abgesetzt werden.

Um mehr als 248 Byte (ohne Multitag), 244 Byte (mit Multitag) und/oder verschiedene Adressbereiche schneller zu lesen oder zu beschreiben, können mehrere Befehle verkettet an das SLG U92 geschickt und im SLG U92 gehalten werden.

Für die Bearbeitung von großen und/oder verschiedenen Datenbereichen können auch verschiedene Funktionen aneinander gehängt (verkettet) werden.

Die Befehle der Befehlskette werden durch ein Bit in den oberen 4 Bit im Befehlsbyte (2. Byte = Byte 1) gekennzeichnet. Das Bit 6 ist auf “1“ gesetzt und die Bits 4, 5 und 7 müssen “0“ sein. Im letzten Befehl der Befehlskette muss das Bit 6 gleich “0“ sein. Dadurch wird das Kettenende signalisiert. Die unteren 4 Bit vom Befehlsbyte enthalten die Funktionskennung. Das heißt, alle aneinander geketteten Befehle müssen vom Befehlstyp 4x hex sein. Der letzte Befehl in einer Kette muss vom Typ 0x hex sein.

Die Befehlskette darf schon an das SLG U92 geschickt werden, wenn der zu bearbeitende MDS sich noch nicht im Feld vom SLG U92 befindet. Sobald der MDS in das Feld eintritt und vom SLG U92 erkannt wird, arbeitet das SLG U92 die verketteten Befehle ab und schickt die Quittungen mit Daten zurück. Wenn während des Sendens der Befehlskette an das SLG U92 bereits die Bearbeitung der Befehlskette begonnen werden kann, so können die ersten Quittungen innerhalb des Sendevorgangs eintreffen.

Achtung

Die maximale Länge einer Befehlskette darf 150 Befehle nicht überschreiten.

Mögliche Verkettungsvarianten:

a) n x WRITE

b) n x READ

c) MDS-Befehle in beliebiger Reihenfolge: INIT, WRITE, READ und/oder MDS-STATUS

d) MDS-Befehle in beliebigerReihenfolge: a), b) oder c) plus END-Befehl

e) RESET-Befehl und nachfolgende Variante a), b), c) oder d)

f) SET-ANT (EIN) plus nachfolgende Variante a), b), c) oder d)

g) Variante a), b), c), d), e) oder f) plus SET-ANT (AUS)

h) Variante a), b), c), d) oder f) plus SET-ANT (AUS) mit SLG-STATUS an beliebiger Position

i) RESET-Befehl plus nachfolgende Variante a), b), c) oder d) mit SLG-STATUS an beliebiger Position nach dem RESET

Nicht zulässige Befehle in der Befehlskette oder unzulässige Verkettung:

- COPY, GET und REPEAT sind in der Befehlskette nicht zulässig (COPY und GET nur mit Multitag)
- RESET innerhalb oder am Ende einer Befehlskette
- END am Anfang oder innerhalb einer Befehlskette
- END ohne vorherigen MDS-Befehl
- SET-ANT (AUS) am Anfang oder vor dem letzten MDS-Befehl oder vor dem END

Beispiel zu verketteten MDS-Befehlen:

- 3 READ-Befehle, gekettet (506 Byte in einem Datenblock)

AB	Befehl	Status	Adresse	Länge	
					1. Befehl
05	42	00	00 00	F8	248 Byte lesen
AB	Befehl	Status	Adresse	Länge	
					2. Befehl
05	42	00	00 F8	F8	248 Byte lesen
AB	Befehl	Status	Adresse	Länge	
					3. Befehl (letzter Befehl)
05	02	00	01 F0	0A	10 Byte lesen

- 3 READ-Befehle, gekettet (3 nicht zusammenhängende Datenblöcke)

AB	Befehl	Status	Adresse	Länge	
05	42	00	00 00	14	1. Befehl 20 Byte lesen
05	42	00	00 F0	1F	2. Befehl 31 Byte lesen
05	02	00	02 01	A5	3. Befehl (letzter Befehl) 165 Byte lesen

Alle MDS-Befehlstypen können gekettet werden. Darüber hinaus kann die Kette mit dem Befehl END abgeschlossen werden. Dieser Befehl darf nicht innerhalb einer Kette stehen.

- INIT-, WRITE- und READ-Befehl, gekettet

AB	Befehl	Status	Datum		Länge	
06	43	00	00	00	80 00	1. Befehl MDS initialisieren

AB	Befehl	Status	Adresse	Länge	Daten	
05	41	00	00 F0	05	31 37 33 39 30	2. Befehl 5 Byte schreiben

AB	Befehl	Status	Adresse	Länge	
05	02	00	02 01	1F	3. Befehl (letzter Befehl) 31 Byte lesen

B.6 Statusbyte status

Der Aufbau des Statusbytes in den Quittungen und Meldungen vom SLG wird nachfolgend mit den möglichen Fehlercodes, die insgesamt bei den Quittungen und Meldungen auftreten können, aufgeführt.

Byte status

Bit	7	6	5	4	3	2	1	0

status Bitmuster Bit 7 bis 6 = 0

Bit 5 = 0 (ECC ist immer eingeschaltet)

Bit 4 bis 0 = Statuscode (00 hex bis 1F hex)

- 00 Kein Fehler;
  Funktion fehlerfrei ausgeführt; Meldung ohne Fehler
- 01 Anwesenheitsfehler: MDS aus dem Feld, wenn Befehl aktiv
- 02 Ein anstehender MDS-Befehl wurde durch den Befehl „Antenne ausschalten“ abgebrochen.
- 03 –
- 04 Fehler im Speicher des MDS
- 05 Unbekannter Befehl /
  falscher oder fehlerhafter Parameter /
  Funktion nicht erlaubt
- 06 Feldstörung am SLG
  > Der MDS hat während der Kommunikation das Feld verlassen.
  > Die Kommunikation zwischen SLG und MDS ist durch äußere Störeinflüsse abgebrochen worden.
- 07 –
- 08 –
- 09 –
- 0A –
- 0B Speicher des MDS ist nicht korrekt lesbar
- 0C Speicher des MDS kann nicht beschrieben werden
- 0D Fehler in der angegebenen Adresse (Adressfehler)
  > Die angegebene Adresse ist auf dem MDS nicht vorhanden
  > Der Befehl ist beim Telegrammaufbau zu überprüfen und zu korrigieren
  > Der MDS ist nicht vom richtigen Typ
- 0E –
- 0F –
- 10 NEXT-Befehl nicht zugelassen

11 –

12 –

13 Im SLG sind nicht genügend Puffer für die Speicherung des Befehls vorhanden.

14 Watchdog-Meldung aus SLG U92

15 Falscher oder fehlerhafter Parameter in der Funktion RESET

16 –

17 –

18 Nur RESET-Befehl zulässig

19 Vorheriger Befehl ist aktiv

1A –

1B Sendeauftrag im SLG zu häufig wiederholt / Datenverlust möglich / RESET-Befehl erforderlich

1C Antenne ist schon ausgeschaltet. / Antenne ist schon eingeschaltet. / Modus im Befehl SET-ANT ist nicht bekannt. / Antenne ist ausgeschaltet, der MDS-Befehl kann nicht ausgeführt werden.

1D Unzulässige Anzahl MDS im Feld
> größer 1
Bei Normaladressierung: MOBY I-aufrufkompatibel (Variante 1 und 2)
> größer Pulkangabe im Befehl RESET

1E Falsche Anzahl Zeichen im Telegramm

1F Laufender Befehl durch RESET-Befehl abgebrochen

C Prozedur 3964R

Die 3964R-Prozedur ermöglicht eine gesicherte Datenübertragung bei einer Punkt-zu-Punkt-Verbindung. Die gesicherte Übertragung wird durch blockweise Übertragung mit Parity, Blockprüfzeichen (BCC) und Empfangsquittung erreicht. Im Datenblock können alle Zeichen von 00 hex bis FF hex übertragen werden.

Zeichenrahmen

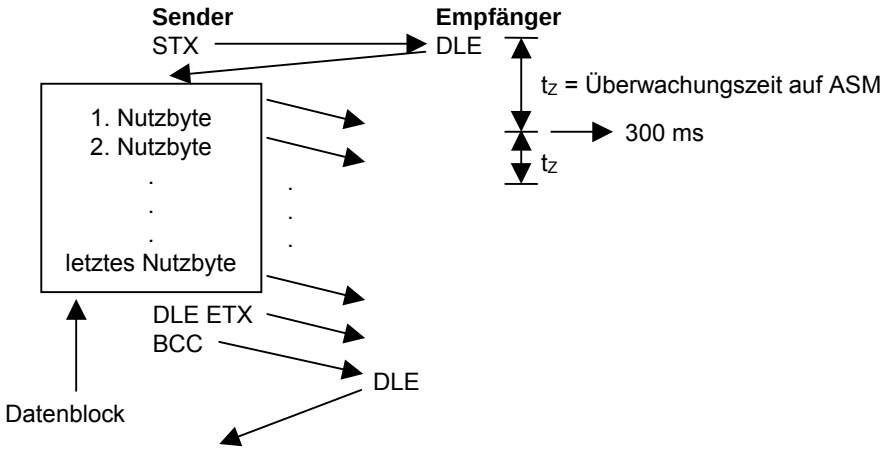
Übertragung:	asynchron
Baudrate:	9600, 19200, 38400, 57600, 115200 Baud
Datenbits:	8
Parität:	odd (ungerade Parität)
Stoppbit:	1

Steuerzeichen in der 3964R-Prozedur

Tabelle C-1 Steuerzeichen in der 3964R-Prozedur

Zeichen	Code (hex)	Bedeutung
STX	02	Initialisierung eines Sendevorgangs (Sendewilligkeit signalisieren)
DLE ETX	10 03	Kennzeichnung des Endes eines Übertragungsblockes
DLE	10	Empfangsbereitschaft (bzw. DLE-Verdopplung im Datenstrom)
NAK	15	Negative Rückmeldung bei Block-Check-Fehler bzw. bei nicht erkanntem Startzeichen
DLE DLE	10 10	DLE-Verdopplung im Datenblock; wird verwendet, wenn der Wert 10 hex im Datenstrom vorkommt

Ablauf einer Blockübertragung



D Begriffe/Abkürzungen, Literaturverzeichnis

D.1 Begriffe/Abkürzungen

ASM	Anschaltmodul
CHN	Kanalnummer
DA	Digitale Ausgänge
DE	Digitale Eingänge
ECC	Error Correction Code
ID	Identifikation
IP	Internet Protocol
MDS	Mobiler Datenspeicher
NAK	Negative Acknowledge
PC	Personal Computer
SIM	Serielles Interface Modul
SLA	Schreib-/Leseantenne
SLG	Schreib-/Lesegerät
SW	Software
TCP/IP	Transmission Control Protocol/Internet Protocol

D.2 Literaturverzeichnis

/01/ Technische Beschreibung Anschaltmodul ASM 420
6GT2 097-3AF00-0DA1

/02/ Technische Beschreibung Serielles Interface Modul SIM
6GT2 097-3AD00-0DA1

/03/ MOBY F Handbuch für Projektierung, Montage und Service
6GT2 497-4BA00-0EA1

/04/ MOBY E Handbuch für Projektierung, Montage und Service
6GT2 397-4BA00-0EA1

/05/ MOBY I Handbuch für Projektierung, Montage und Service
6GT2 097-4BA00-0EA1

/06/ MOBY U Handbuch für Projektierung, Montage und Service
6GT2 597-4BA00-0EA1

/07/ MOBY-Benutzerhandbuch 3964R-Protokoll unter Windows NT 4.0/95
(auf Diskette MOBY-Software enthalten)