SIEMENS

MOBY®

MOBY API C-Library

Programming Instructions

Published in March 2005

Safety Guidelines

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring to property damage only have no safety alert symbol. These notices shown below are graded according to the degree of danger.

⚠ **Danger**

indicates that death or severe personal injury **will** result if proper precautions are not taken.

⚠ **Warning**

indicates that death or severe personal injury **may** result if proper precautions are not taken.

⚠ **Caution**

with a safety alert symbol, indicates that minor personal injury can result if proper precautions are not taken.

Caution

without a safety alert symbol, indicates that property damage can result if proper precautions are not taken.

Notice

indicates that an unintended result or situation can occur if the corresponding information is not taken into account.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified personnel

The device/system may only be set up and used in conjunction with this documentation. Commissioning and operation of a device/system may only be performed by **qualified personnel**. Within the context of the safety notes in this documentation qualified persons are defined as persons who are authorized to commission, ground and label devices, systems and circuits in accordance with established safety practices and standards.

Trademarks

All names identified by ® are registered trademarks of the Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Siemens-Aktiengesellschaft

Order no. (4)J31069-D0137-U001-A5-7618

Table of Contents

1 Introduction

1.1 Product Overview

The MOBY family of products includes a number of identification systems which are used throughout the world to control and optimize material flow in production, manufacturing, distribution and logistics.

The identification systems are made up of three components.

- Mobile data memories (MDS)
- Read/write devices (SLG), read/write antennas (SLA) or serial interface modules (SIM)
- Interface modules (ASM)

These components are available in various models.

A uniform MOBY API C-library is available for user-specific MOBY applications for communication with the MOBY systems which are linked serially to PC with Windows 98/NT 4.0/2000/XP or to Ethernet with PC with Windows 98/NT 4.0/2000/XP. It offers a simple and quick system link.

1.2 Serial Link to PC

The serial link of the MOBY systems to the PC with Windows 98/NT 4.0/2000/XP is provided by ASM, SIM or SLG as shown in table 1-1.

The MOBY components are usually equipped with physical interfaces – RS 232 for short cables and RS 422 for long cables. The protocol is always 3964R. See appendix C.

1.2.1 MOBY Systems for Serial Connection

Table 1-1 Components which can be run with MOBY API with serial link to PC

MOBY Product Family	Interface	Documentation
MOBY E	ASM 420 with SLG 7x	/01/
	ASM 424 with SLG 7x (one to four SLG 7x on one ASM 424)	/05/
	ASM 724 with SLA 71 (one to four SLA 71 on one ASM 724)	/04/
MOBY F	ASM 824 with SLA 81 (one to four SLA 81 on one ASM 824)	/03/
MOBY I (without filehandler)	ASM 420 with SLG 4x	/01/
	ASM 424 with SLG 4x (one to four SLG 4x on one ASM 424)	/05/
	SIM 41	/02/
MOBY U (without filehandler)	SLG U92	/06/

1.2.2 C-library - MOBY API for Serial Link to PC

The MOBY API C-library for user-specific MOBY applications runs under Windows 98/NT 4.0/2000/XP. It implements and runs MOBY applications from the following components.

• C-library	MOBY_API.LIB
• MOBY header file	MOBY_API.H
• Dynamic link library	MOBY_API.DLL
• 3964R driver	3964R.DLL
• Configuration program for 3964R	CPL3964R.CPL
• 3964R header file	3964R.H
• Sample application	
– Executable	EXAMPLE.EXE
– Source code	EXAMPLE.CPP

The read/write devices (SLG), read/write antennas (SLA) and the serial interface modules (SIM) are controlled via the serial interface by transmitting structured telegrams with the 3964R protocol. The 3964R protocol is a DLL (Dynamic Link Library).
The telegrams are generated or received by library functions. The library functions are based on the 3964R driver and provide the interface between MOBY applications and MOBY systems.

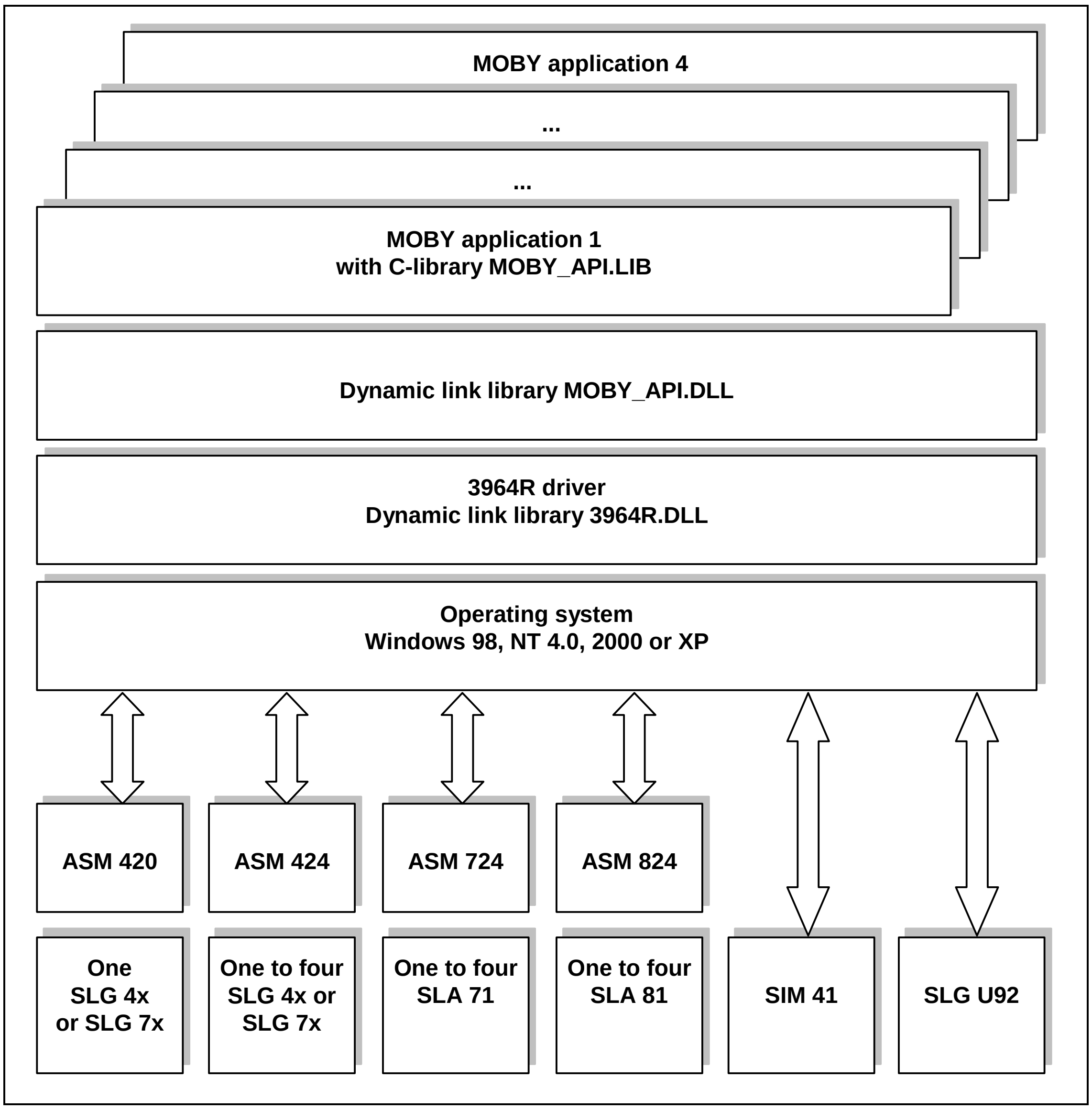


Figure 1-1 Structure of MOBY API for serial link to PC

The functions are available as DLL in the dynamic link library (MOBY_API.DLL) and the function calls in the C-library (MOBY_API.LIB).

- Maximum of one application per ASM, SIM, or SLG U92
- One link library for up to four applications

The driver functions must be configured as a CPL file under the system controller of Windows.

• Configuration program	CPL3964R.CPL

A header file with function declarations must be linked to each of the MOBY applications.

• Header file	MOBY_API.H

The MOBY_API.H header file must be linked to the source code of the application with preprocessor command "#include." This declares all function calls and constants. The header file was developed with and for Visual C++. If you want to use another programming language or a C++ dialect from another company, the header file may have to be adjusted.
The MOBY_API.H header file requires the include file (3964R.H) from the 3964R driver.

Sample application

In addition to the C-library, an executable sample application (EXAMPLE.EXE) is supplied which makes it easy to get started with the implementation of a user application. It is also available in source code (EXAMPLE.CPP). It can be used as the basis of your own user application which can be fully linked in and executed.

1.2.3 System Prerequisites for Serial Link to PC

The following prerequisites must be met before the C-library can be used under Windows^{TM}.

• Personal computer (PC)	AT-compatible PC or PG
• Operating system	Windows^{TM} 98/NT 4.0/2000/XP
• Free serial interface(s)	RS 232/RS 422
• Programming regulations	The MOBY_API.LIB interface library is written in "C" and is compatible with Microsoft Visual C++ compiler versions ≥ 6.0. Other programming languages (e.g., Visual Basic) with a wrapper.

1.3 Link to Ethernet

MOBY systems are linked to Ethernet with PC with Windows 98/NT 4.0/2000/XP via the ASM 480 interface module as shown in Table 1-2. The MOBY system is connected serially to the ASM 480. The protocol between the ASM 480 and the MOBY system is always 3964R. The ASM 480 is used as the protocol converter between the TCP/IP protocol and the 3964R protocol.

1.3.1 MOBY Systems Which Can Be Connected to Ethernet

Table 1-2 Components which can be used with MOBY API with coupling to Ethernet

MOBY Product Family	Interface	Related Documentation
MOBY U (without filehandler)	ASM 480 with SLG U92	/06/

1.3.2 MOBY API C-Library for Link to Ethernet

The MOBY API C-library product for user-specific MOBY applications can be executed under Windows 98/NT 4.0/2000/XP. It is used for the implementation and execution of MOBY applications with the following components.

• C-library	MOBY_API_T.LIB
• MOBY header file	MOBY_API_T.H
• Dynamic link library	MOBY_API_T.DLL
• Sample applications	
– Executable (stored under EXAMPLE.ZIP, subdirectory "MOBY_API_T")	MOBY API.EXE

– Source code of the executable sample application MOBY API.EXE
(stored under EXAMPLE.ZIP, subdirectory "MOBY_API_T")

– Source code for a simple sample application
(stored under EXAMPLE.ZIP, subdirectory "TEST")

Control of the read/write devices (SLGs) is handled on Ethernet via transmission of structured telegrams which are transferred to the serial interface of the SLG with the 3964R protocol.
The telegrams are generated or received via library functions. The library functions are based on the TCP/IP driver (dynamic link library) and provide the interface between the MOBY applications and the MOBY systems.

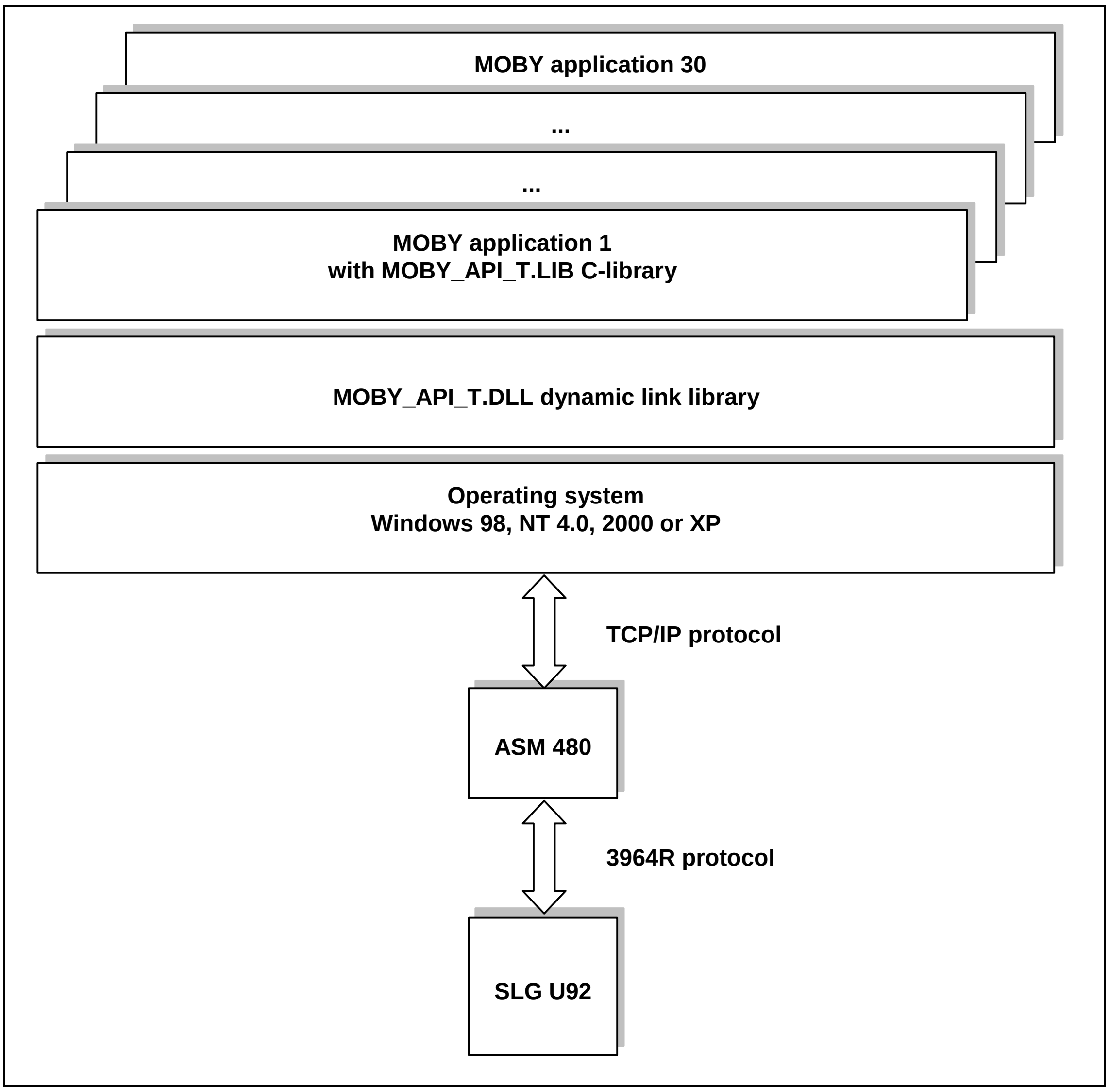


FIgure 1-2 MOBY API structure with link to Ethernet

The functions are available as DLL (Dynamic Link Library) in the MOBY_API_T.DLL dynamic link library and the function calls in the MOBY_API_T.LIB C-library.

- Maximum of one application per SLG U92
- One link library for a maximum of 30 applications

A header file with function declarations must be integrated in each MOBY application.

- Header file MOBY_API_T.H

The header file MOBY_API_T.H must be integrated in the source code of the application with the preprocessor command "#include." This declares all function calls and constants. The header file was developed with and for Visual C++. If you want to use a different programming language or a C++ dialect of another company, you may have to adjust the header file.

Sample applications

In addition to the C-library, two sample applications are included for an easy introduction to implementation of a user application.

- Executable application MOBY API.EXE for control of up to four SLG U92 devices with ASM 480 which is also available in source code
- Simple sample application TEST.CPP in source code (VC++6.0) as the basis for creating your own user application for an SLG U92 with ASM 480. It gives you an overview of the primary MOBY API functions.

  For more information, see also chapter 5.2.

Both sample applications are included in the EXAMPLE.ZIP file.

1.3.3 System Prerequisites for Link to Ethernet

Use of the C-library under Windows™ requires the following prerequisites.

• Personal Computer (PC)	AT compatible PC or PG
• Operating system	Windows^{TM} 98/NT 4.0/2000/XP
• LAN connection to PC	
• Programming rules	The interface library MOBY_API_T.LIB is written in "C" and is compatible with Microsoft Visual C++ compiler version ≥ 6.0. Other programming languages (e.g., Visual Basic) via a wrapper

2 Installation

2.1 Files Supplied

• mobyapi.msi	Windows Installer Package with C-library for MOBY applications for serial linking to PC. Use the Windows installation package (mobyapi.msi) to install/deinstall the C-library for MOBY applications.
• mobyapi_t.msi	Windows Installer Package with C-library for MOBY applications for linking to Ethernet. You can use the Windows installation package (mobyapi_t.msi) to install or de-install the C-library for MOBY applications.
• InstMsiW.exe	Windows Installer for Windows NT
• InstMsiA.exe	Windows Installer for Windows 98

Notice

If the Windows installation package mobyapi.msi or mobyapi_t.msi with Windows NT or Windows 98 is not indicated as type "Windows Installer Package," the Windows Installer must be activated.

The installation of the C-library MOBY API is executed in its own installation sector depending on the link (interface version).

• Installation for serial link to PC:	mobyapi.msi
• Installation for link to Ethernet:	mobyapi_t.msi

2.2 Installation of the C-Library - MOBY API for Serial Link to PC

After starting the Windows installation package (mobyapi.msi), you will be guided through installation.

1. Start the mobyapi.msi package.

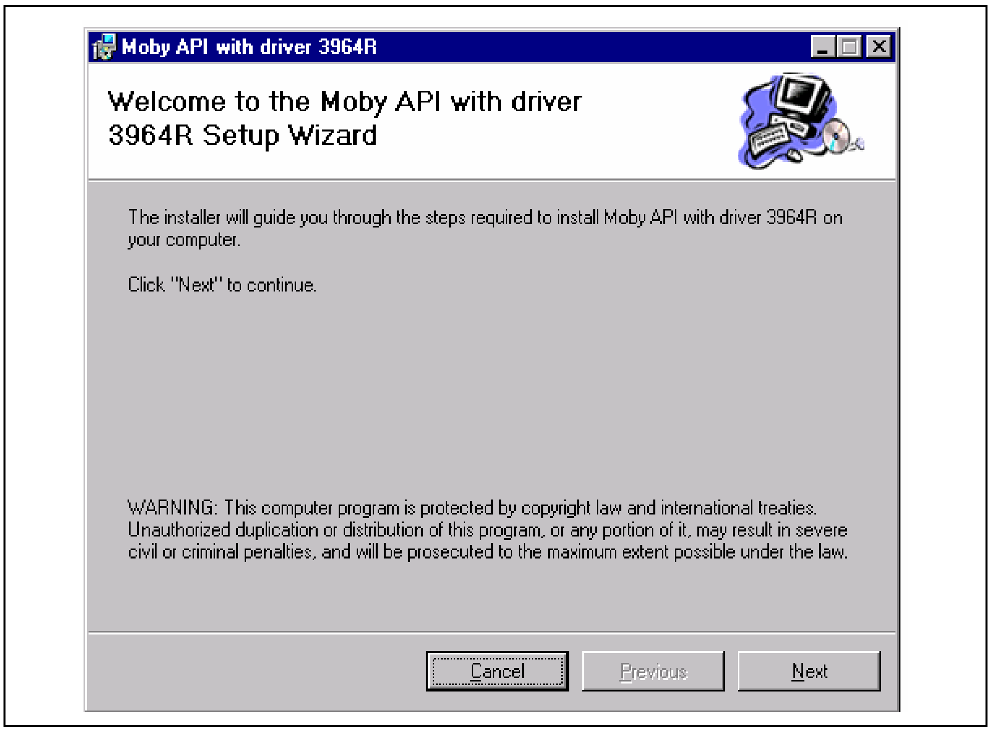
   The first time the C-library is installed, the "Welcome to the MOBY API ...“ screen appears (see figure 2-1). Otherwise the "Select whether you want to repair or remove MOBY API ...“ screen is opened (see figure 2-6).

Notice

The MOBY API C-library should never be partially or completely installed/de-installed manually. This may cause problems during installation/deinstallation with mobyapi.msi.

New installation

2. Activate the installation of the C-library


Moby API with driver 3964R

Welcome to the Moby API with driver 3964R Setup Wizard

The installer will guide you through the steps required to install Moby API with driver 3964R on your computer.

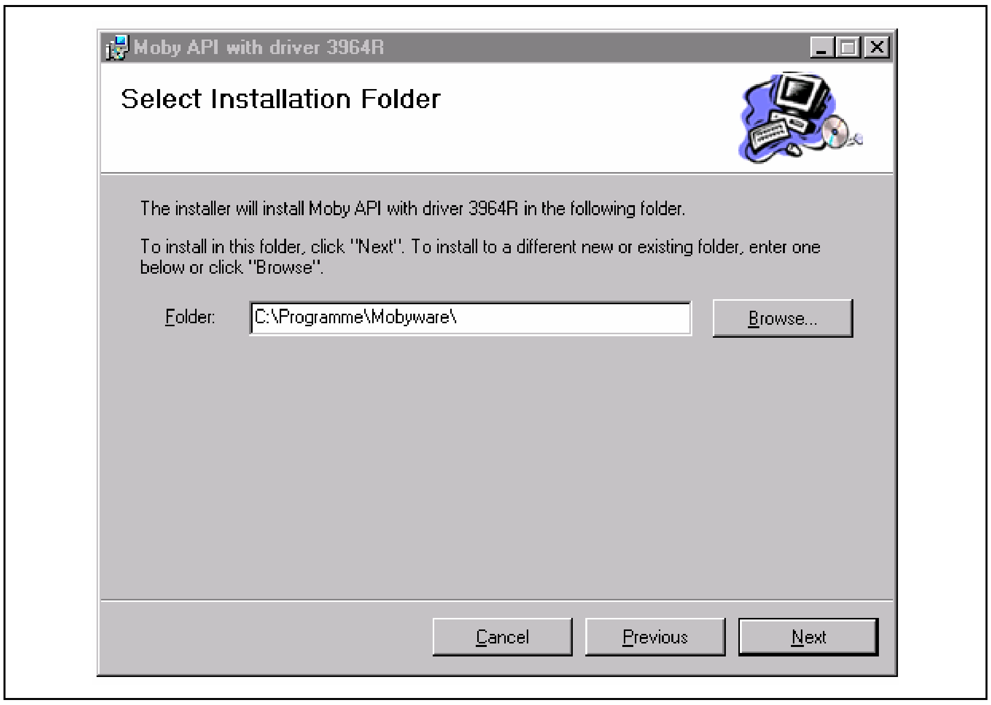
Click "Next" to continue.

WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.

Figure 2-1 Activating installation of the C-library (serial link to PC)

Click "Next" to activate installation. The screen "Select Installation Folder“ appears (see figure 2-2).
"Cancel" terminates installation (with previous confirmation).

3. Select directory for the C-library MOBY API.



Moby API with driver 3964R

Select Installation Folder

The installer will install Moby API with driver 3964R in the following folder.

To install in this folder, click "Next". To install to a different new or existing folder, enter one below or click "Browse".

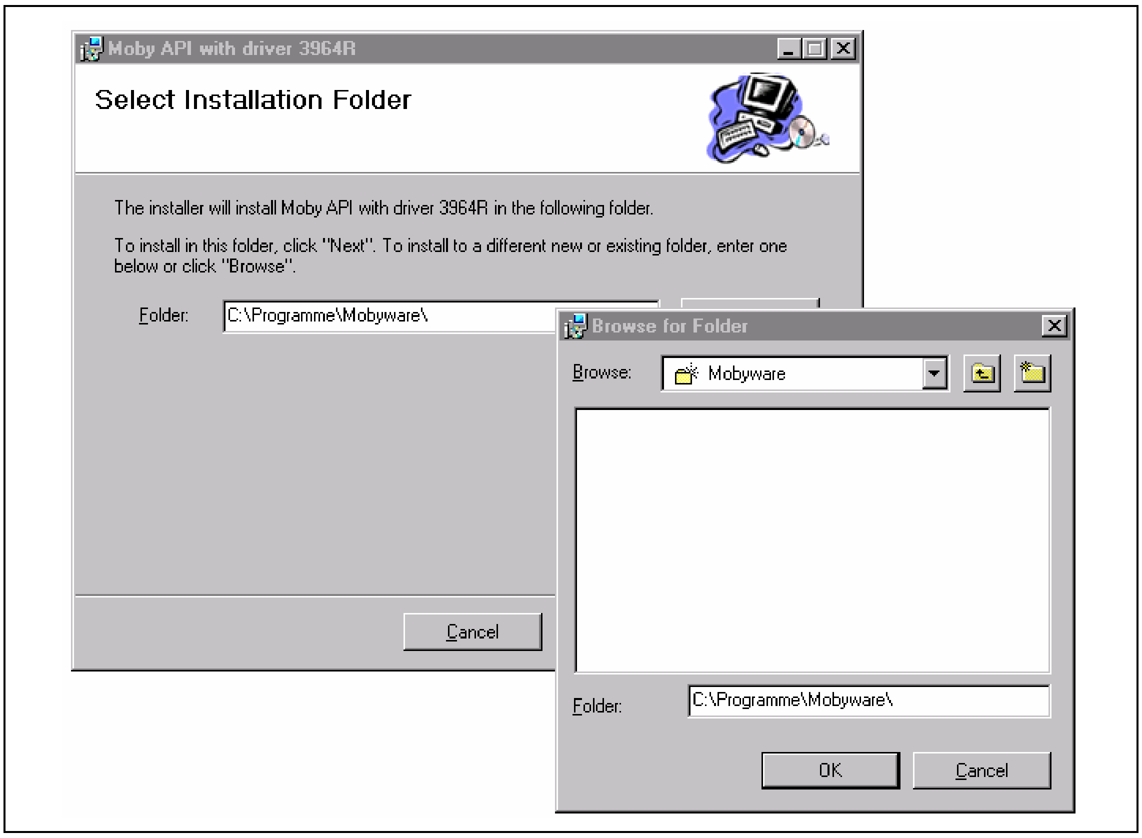
Folder: C:\Programme\Mobyware\

Figure 2-2 Selecting directory for installation (serial link to PC)

The screen "Select Installation Folder“ appears with the standard setting "C:\Programme\mobyware." Press "Next" to accept this setting. Use the "Folder" field to change the directory or the "Browse" field to select an existing one (see figure 2-3).

"Next" accepts the setting, and the screen "Confirm Installation“ appears (see figure 2-4).

4. Select an existing directory.


Moby API with driver 3964R

Select Installation Folder

The installer will install Moby API with driver 3964R in the following folder.

To install in this folder, click "Next". To install to a different new or existing folder, enter one below or click "Browse".

Folder: C:\Programme\Mobyware\

Browse for Folder

Browse: Mobyware

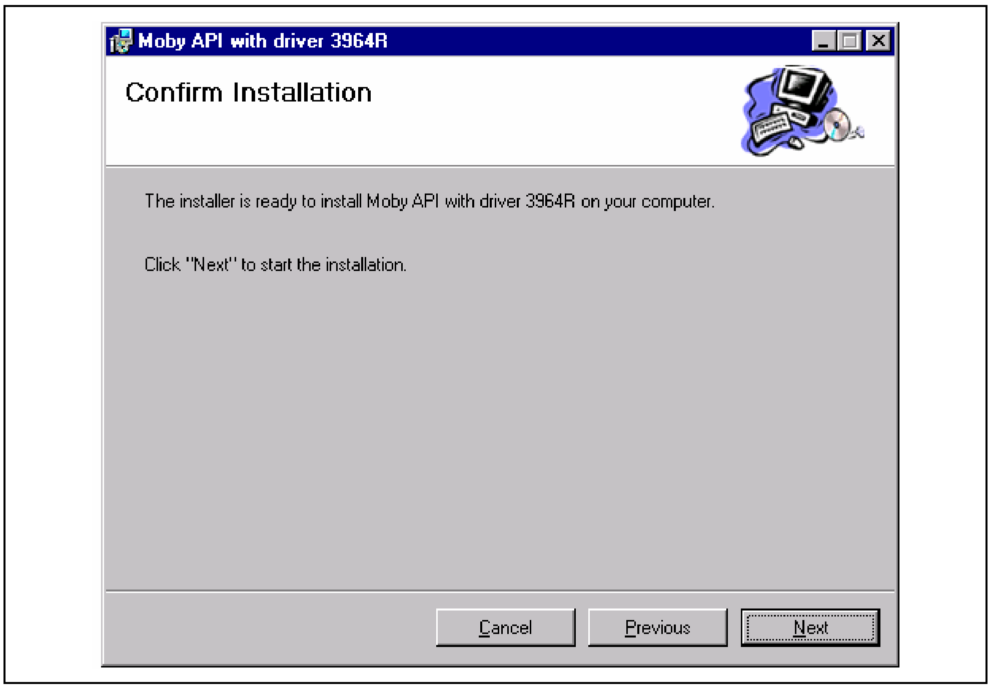
Folder: C:\Programme\Mobyware\

Figure 2-3 Selecting existing directory (serial link to PC)

You can select an existing directory in the "Browse for Folder" screen. An existing directory can be accepted or changed in the "Folder" field.

- OK returns you to the "Select Installation Folder" screen with the acceptance of the directory.
- Cancel also returns you. Your selection is cancelled.

5. Execute installation of MOBY API software.



The installer is ready to install Moby API with driver 3964R on your computer.

Click "Next" to start the installation.

Figure 2-4 Executing installation (serial link to PC)

Press "Next" to store the MOBY API software for the creation of applications under the following subdirectories within the selected directory.

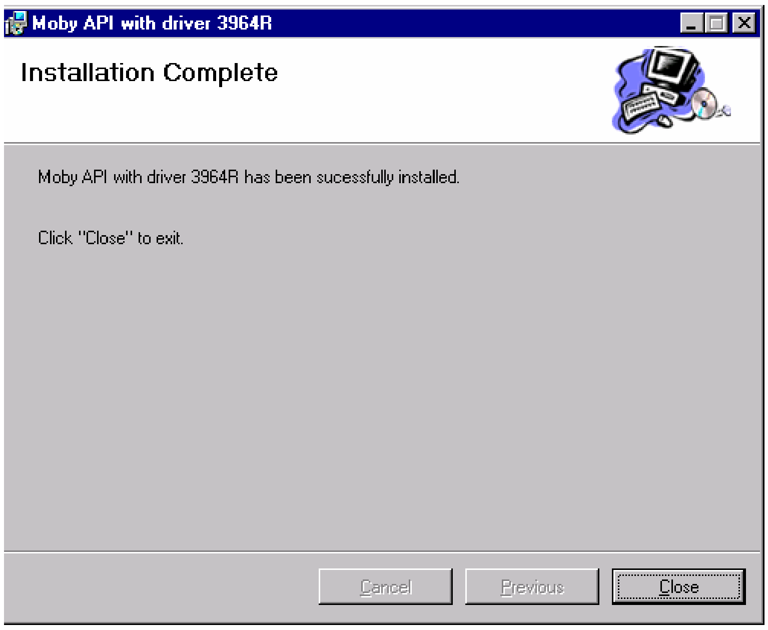
\example	example.cpp	Sample application in source code
	example.exe	Sample application as executable program
\include	3964R.H	Include file of 3964R driver for MOBY_API.H header file
	MOBY_API.H	Header file for user applications
\lib	MOBY_API.LIB	C-library for user applications

The DLL of the 3964R driver and the DLL MOBY_API and the CPL3964R.CPL configuration program are copied to the following directory (based on the Windows system).

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

The 3964R driver must be configured before the sample application can be used or a user application can be tested (see chapter 2.2.2). The driver DLL contains information on existing interfaces and their configuration using the Windows registry. The entries in the registry for use of the 3964R driver are generated automatically during installation.

6. Installation of the existing MOBY API C-library is complete.



Moby API with driver 3964R has been sucessfully installed.

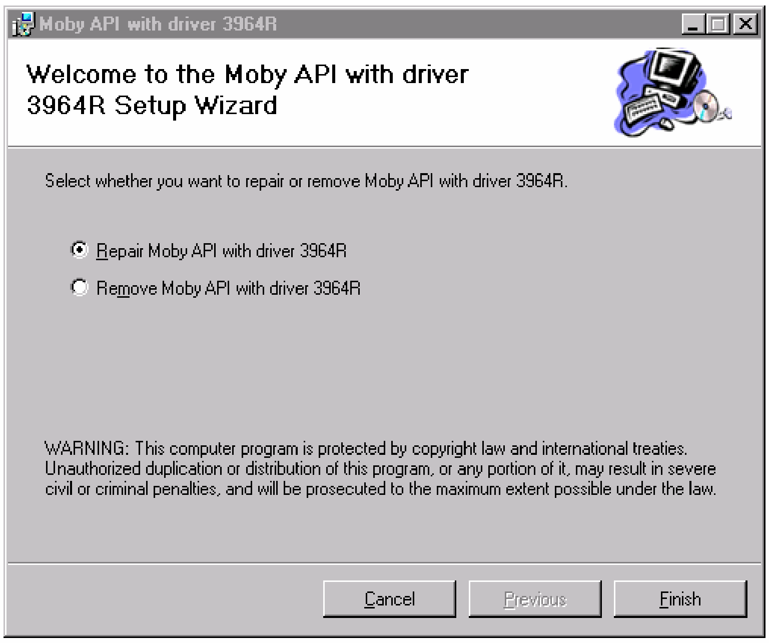
Click "Close" to exit.

Figure 2-5 Installation completed (serial link to PC)

Use "Close" to conclude the dialog.

Overwriting an existing C-library

7. Overwrite or deinstall an existing MOBY API C-library.



Moby API with driver 3964R

Welcome to the Moby API with driver 3964R Setup Wizard

Select whether you want to repair or remove Moby API with driver 3964R.

[✓ Repair Moby API with driver 3964R]
Remove Moby API with driver 3964R

WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.

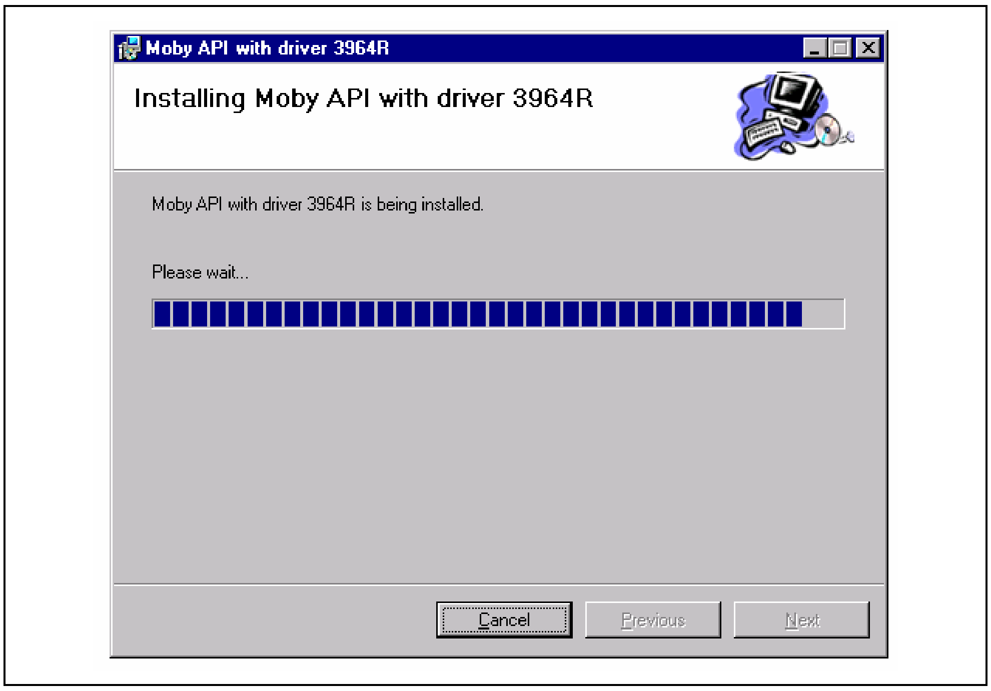
Figure 2-6 Overwriting existing C-library (serial link to PC)

To overwrite the existing MOBY API C-library, select "Repair MOBY API with driver 3964R," and then start the procedure with Finish (see figure 2-7).

Notice

Be sure to add the new header files (3964R.H and MOBY_API.H) and the C-library (MOBY_API.LIB) to the development package.

8. Existing MOBY API C-library is overwritten.


Moby API with driver 3964R

Installing Moby API with driver 3964R

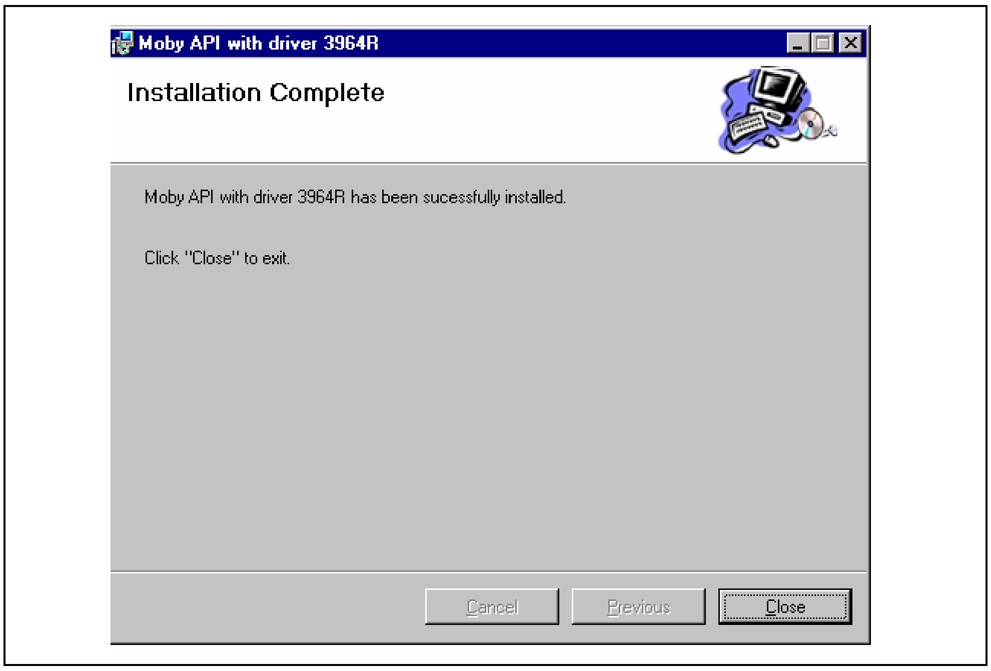
Moby API with driver 3964R is being installed.

Please wait...

Figure 2-7 Screen shown during installation (serial link to PC)

The existing MOBY API C-library is overwritten.

9. Overwriting of the existing MOBY API C-library is complete.



Moby API with driver 3964R has been sucessfully installed.

Click "Close" to exit.

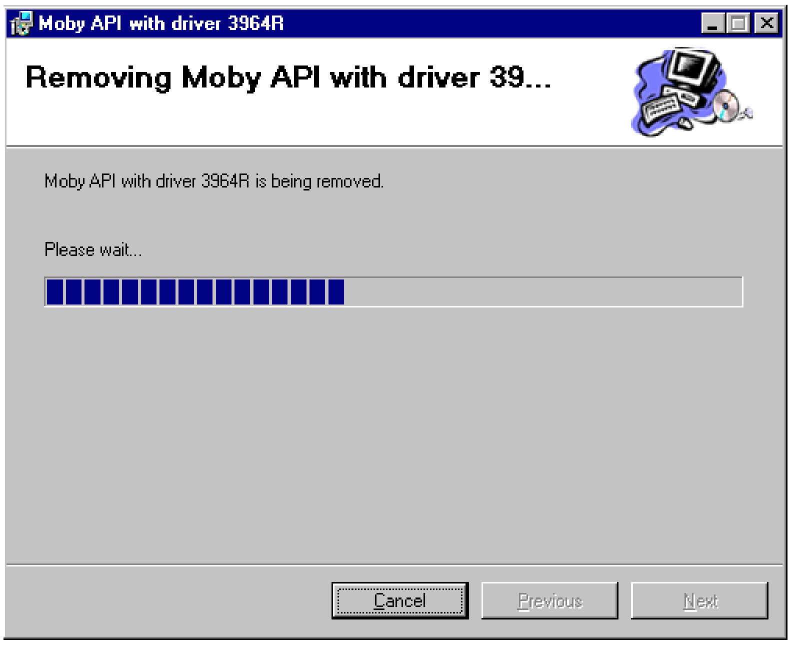
Figure 2-8 Overwriting completed (serial link to PC)

Use Close to conclude the dialog.

Deinstallation

To deinstall the existing MOBY API C-library, select "Remove MOBY API with driver 3964R" (see figure 2-6), and then start the procedure with Finish (see figure 2-9).

10.Deinstall existing MOBY API C-library.



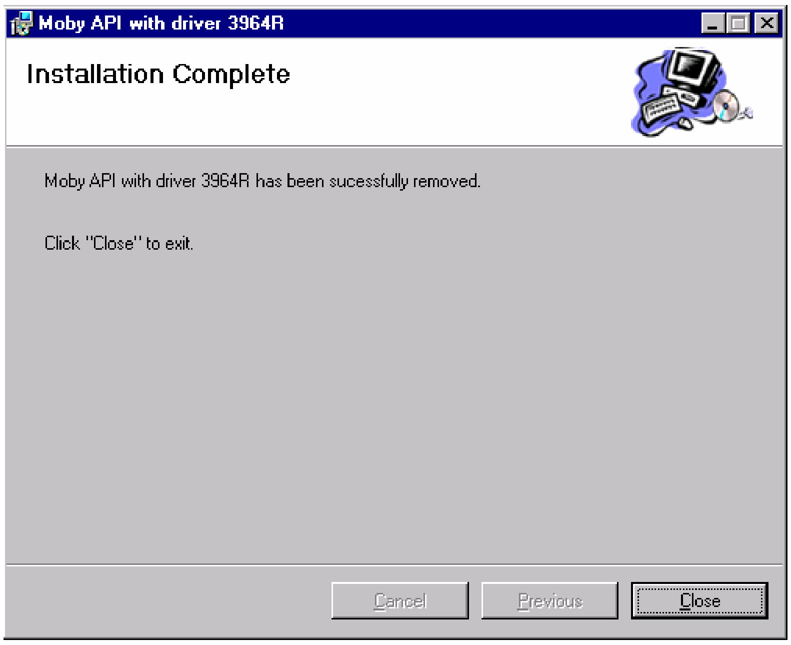
Removing Moby API with driver 39...

Moby API with driver 3964R is being removed.

Please wait...

Figure 2-9 Screen during deinstallation (serial link to PC)

11.Deinstallation of existing MOBY API C-library is complete.



Moby API with driver 3964R has been sucessfully removed.

Click "Close" to exit.

Figure 2-10 Deinstallation is completed (serial link to PC)

Use Close to conclude the dialog.

2.2.1 Adjusting the Registry

Oddity of Windows 2000, Windows NT and Windows XP: If the driver was installed by the administrator/generator-owner and a user wants the rights to the 3964R driver, the following steps must be performed by the administrator/generator-owner.

1. Start the REGEDT32.EXE program.
2. Set the path "Siemens-741" in the HKEY_LOCAL_MACHINE screen (see figure 2-11).

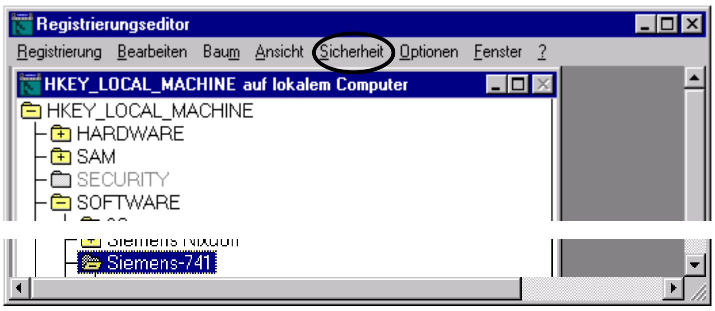
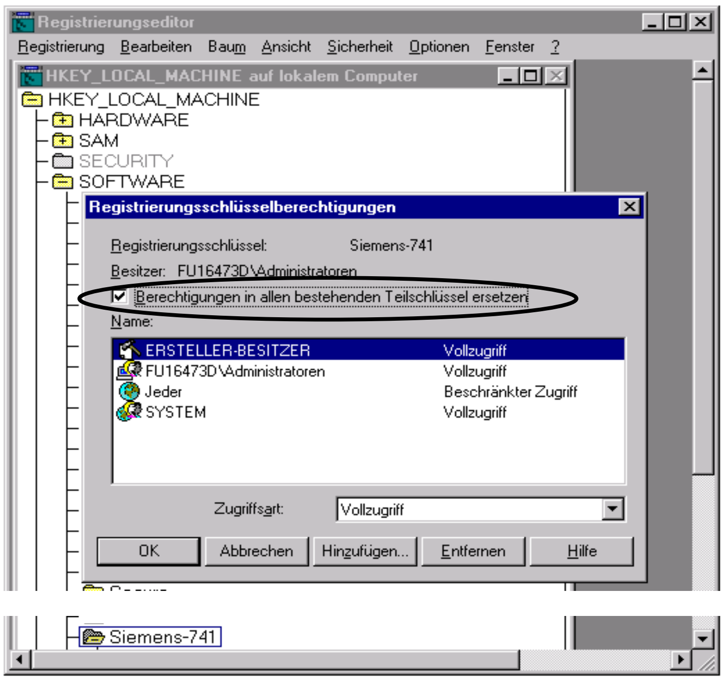

Registrierungseditor
Registrierung Bearbeiten Baum Ansicht Sicherheit Optionen Fenster ?
HKEY_LOCAL_MACHINE auf lokalem Computer
HKEY_LOCAL_MACHINE
HARDWARE
SAM
SECURITY
SOFTWARE
Siemens Nixdorf
Siemens-741

Figure 2-11 The HKEY_LOCAL_MACHINE screen

3. Select "Sicherheit" in the menu bar (see figure 2-11). The "Registrierungsschlüsselberechtigungen" screen appears. In this screen, activate the option "Berechtigungen..." and select the user for whom the rights are to be added (see figure 2-12).


Registrierungseditor
Registrierung Bearbeiten Baum Ansicht Sicherheit Optionen Fenster ?
HKEY_LOCAL_MACHINE auf lokalem Computer
HKEY_LOCAL_MACHINE
HARDWARE
SAM
SECURITY
SOFTWARE
Siemens-741
Registrierungsschlüsselberechtigungen
OK
Abbrechen
Hinzufügen...
Entfernen
Hilfe

Registrierungsschlüssel: Siemens-741
Besitzer: FU16473D\Administratoren
☑ Berechtigungen in allen bestehenden Teilschlüssel ersetzen
Name:

ERSTELLER-BESITZER	Vollzugriff
FU16473D\Administratoren	Vollzugriff
Jeder	Beschränkter Zugriff
SYSTEM	Vollzugriff

Zugriffsart: Vollzugriff

Figure 2-12 "Registrierungsschlüsselberechtigungen" screen

4. Click the "Hinzufügen..." button. In the screen which appears, change "Zugriffsart" to "Vollzugriff" (see figure 2-13). Then close all screens with OK.

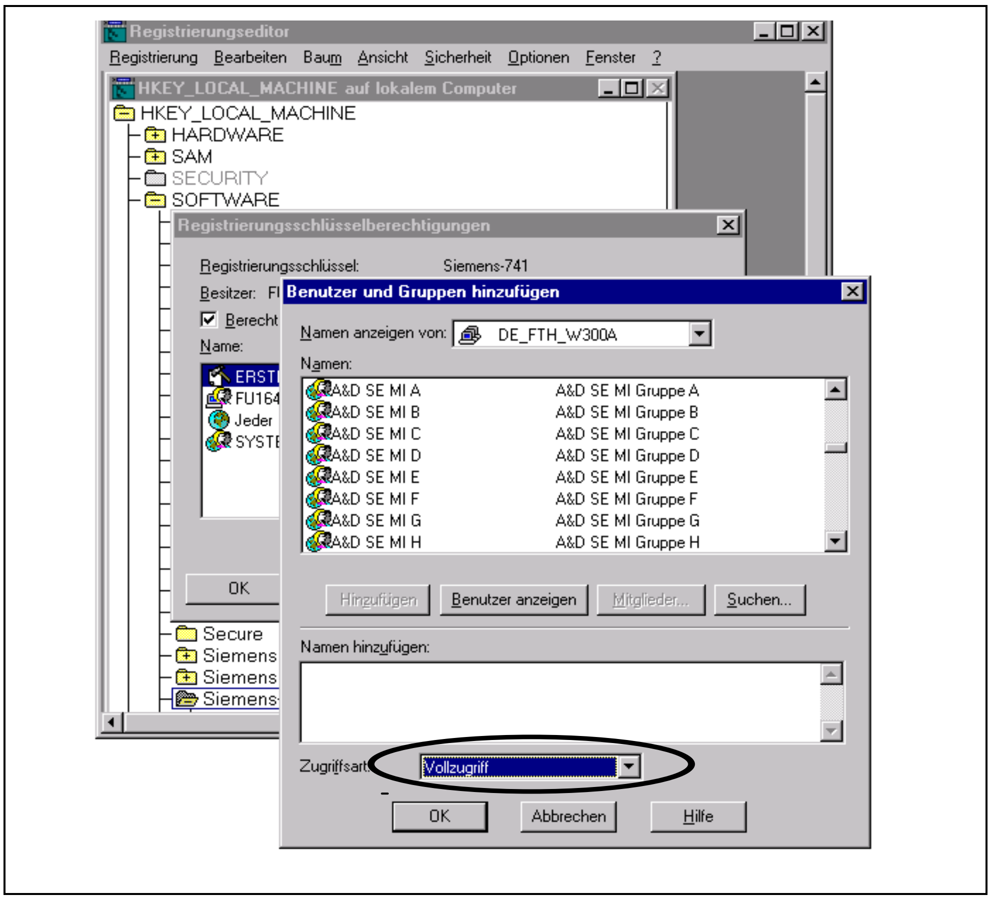

Registrierungseditor
Registrierung Bearbeiten Baum Ansicht Sicherheit Optionen Fenster ?
HKEY_LOCAL_MACHINE auf lokalem Computer
HKEY_LOCAL_MACHINE
HARDWARE
SAM
SECURITY
SOFTWARE
Registrierungsschlüsselberechtigungen
Registrierungsschlüssel: Siemens-741
Besitzer: FI
Berecht
Name:
ERSTI
FU164
Jeder
SYSTE
OK
Secure
Siemens
Siemens
Siemens
Benutzer und Gruppen hinzufügen
Namen anzeigen von: DE_FTH_W300A
Namen:
A&D SE MI A A&D SE MI Gruppe A
A&D SE MI B A&D SE MI Gruppe B
A&D SE MI C A&D SE MI Gruppe C
A&D SE MI D A&D SE MI Gruppe D
A&D SE MI E A&D SE MI Gruppe E
A&D SE MI F A&D SE MI Gruppe F
A&D SE MI G A&D SE MI Gruppe G
A&D SE MI H A&D SE MI Gruppe H
Hinzufügen
Benutzer anzeigen
Mitglieder...
Suchen...
Namen hinzufügen:
Zugriffsart: Vollzugriff
OK
Abbrechen
Hilfe

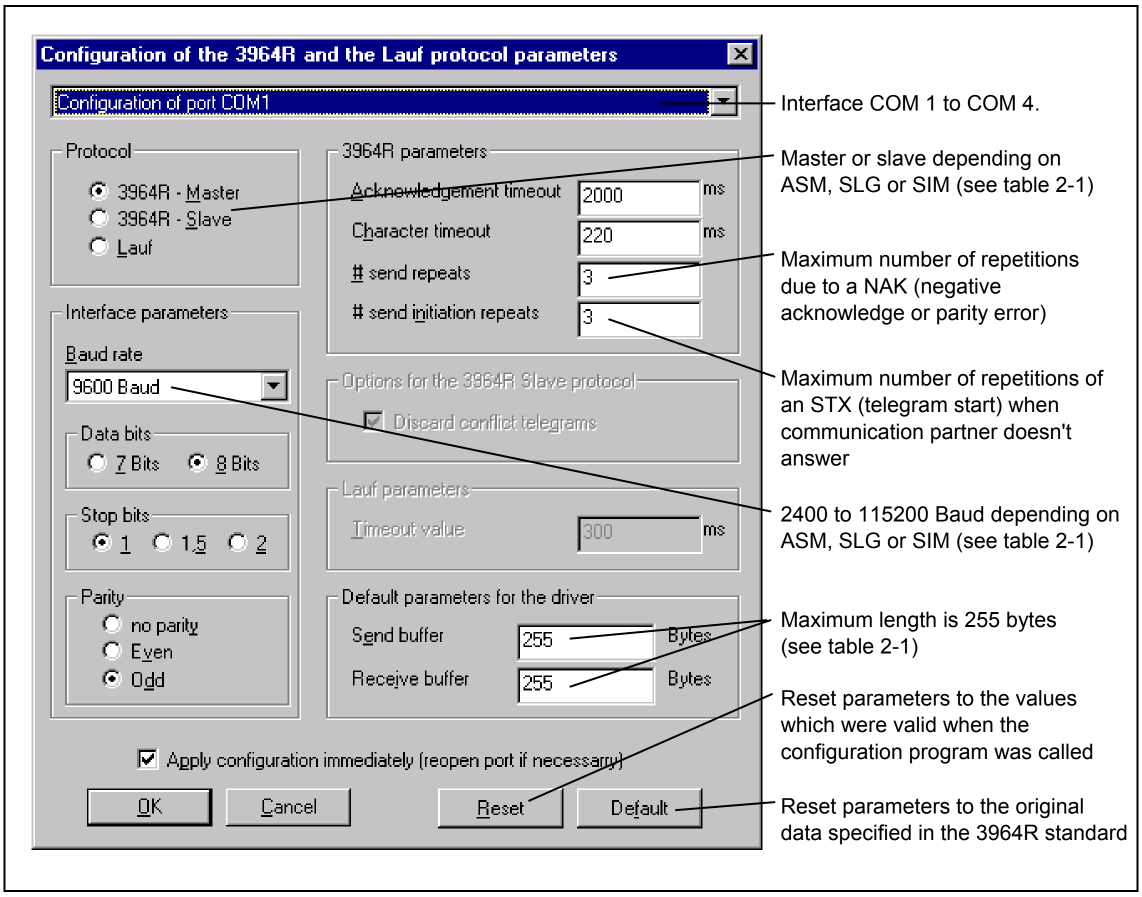
Figure 2-13 "Benutzer und Gruppen hinzufügen" screen

2.2.2 Configuration of the 3964R Driver

The 3964R driver is configured with the CPL3964R.CPL configuration program (see figure 2-14). It must be called under the system control of Windows.

The following parameters must be set as standard values with the configuration program.

• Data bits	8	
• Stop bits	1	
• Parity	Odd	
• Send buffer	255	
• Receive buffer	255	
• Discard conflict telegrams	√	(if 3964R driver is slave)



Configuration of the 3964R and the Lauf protocol parameters

Configuration of port COM1

Protocol: [✓ 3964R - Master] 3964R - Slave Lauf

3964R parameters
- Acknowledgement timeout: 2000 ms
- Character timeout: 220 ms
- # send repeats: 3
- # send initiation repeats: 3

Interface parameters
- Baud rate: 9600 Baud
- Data bits: 7 Bits [✓ 8 Bits]
- Stop bits: [✓ 1] 1.5 2
- Parity: no parity Even [✓ Odd]

Options for the 3964R Slave protocol
- [✓ Discard conflict telegrams]

Lauf parameters
- Timeout value: 300 ms

Default parameters for the driver
- Send buffer: 255 Bytes
- Receive buffer: 255 Bytes

[✓ Apply configuration immediately (reopen port if necessary)]

OK | Cancel | Reset | Default

Interface COM 1 to COM 4.

Master or slave depending on ASM, SLG or SIM (see table 2-1)

Maximum number of repetitions due to a NAK (negative acknowledge or parity error)

Maximum number of repetitions of an STX (telegram start) when communication partner doesn't answer

2400 to 115200 Baud depending on ASM, SLG or SIM (see table 2-1)

Maximum length is 255 bytes (see table 2-1)

Reset parameters to the values which were valid when the configuration program was called

Reset parameters to the original data specified in the 3964R standard

Figure 2-14 Dialog for configuration of the 3964R protocol

Notice

With MOBY U the standard parameterization of “Acknowledgement timeout” is 150 msec and “Character timeout” is 50 msec.

Notice

If the checkbox "Apply configuration immediately" is checked, all data changes are applied immediately in the configuration even when the port needs to be closed and opened (e.g., when the baud rate changes). Otherwise, only the data which do not affect interface parameters are accepted (e.g., timeout values).

Table 2-1 SLG, ASM or SIM-dependent driver parameters

ASM or SIM Dependent Driver Parameter	**SLG/ASM/SIM**					
	ASM 824	**ASM 724**	**ASM 424**	**ASM 420**	**SIM 41**	**SLG U92**
Protocol	3964R slave	3964R slave	3964R slave	3964R master or slave	3964R master or slave	3964R slave
Baud rate	9600, 19200 or 38400 baud			2400, 4800, 9600, 19200 or 38400 baud	2400, 4800 or 9600 baud	19200, 38400, 57600 or 115200 baud
Send buffer	200 bytes	242 bytes	242 bytes	255 bytes	255 bytes	255 bytes
Receive buffer						

The size of the send buffer and the receive buffer is determined by the telegram header and the command-specific data.

Telegram header length:

- 4 bytes with ASM 824, ASM 724 and ASM 424 (see appendix A.5)
- 3 bytes with SLG U92, ASM 420 and SIM 41

Notice

The MOBY user data may not exceed 248 bytes for the data transmission. This means that the maximum length of the data to be read from the MDS or to be written to the MDS is 248 bytes (see chapter 3.4).

2.2.3 Priority Assignment for the 3964R Driver

The Windows operating system may not exceed the times "Acknowledgment timeout" and "Character timeout" during communication between the PC and the SLG, ASM or SIM. Otherwise communication is terminated and started again. Depending on the CPU load and data memory accesses this may affect communication. This means that the above times are exceeded. To bypass or minimize this problem the thread priority for communication is set to 2 when the MOBY API C interface is installed.

Thread priority 2 means:	Maximum priority only during communication This setting is made for every COM interface used by the 3964R driver before the COM interface is opened.

If you reset the thread priority to its original value or you want to change it, make the setting in the registry with the following key.

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

2.3 Installation of the MOBY API C-Library for Link to Ethernet

After the Windows installation package (mobyapi_t.msi) starts, you are guided through installation.

1. Starting the mobyapi_t.msi package

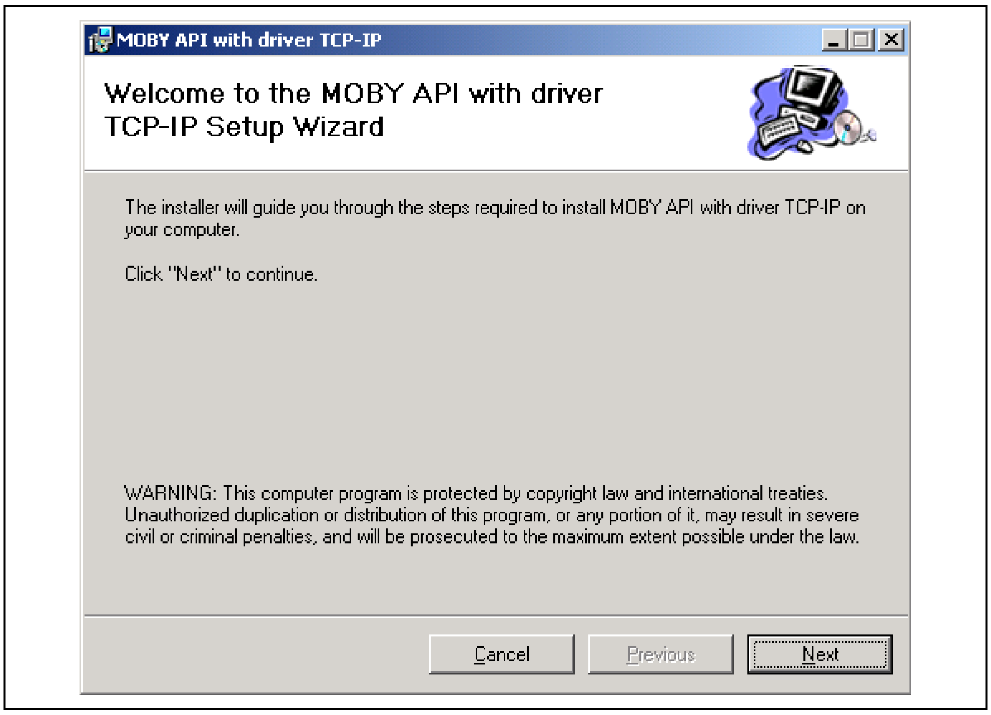
   The first time the C-library is installed, the screen for activating the C-library installation (figure 2-15) appears. Otherwise the screen for overwriting and de-installing an existing MOBY API C-library (figure 2-20) appears.

Notice

The MOBY API C-library should never be manually installed or de-installed either partially or completely. This might cause problems during the installation or de-installation with mobyapi_t.msi.

New installation

2. Activating installation of the C-library



Welcome to the MOBY API with driver TCP-IP Setup Wizard

The installer will guide you through the steps required to install MOBY API with driver TCP-IP on your computer.

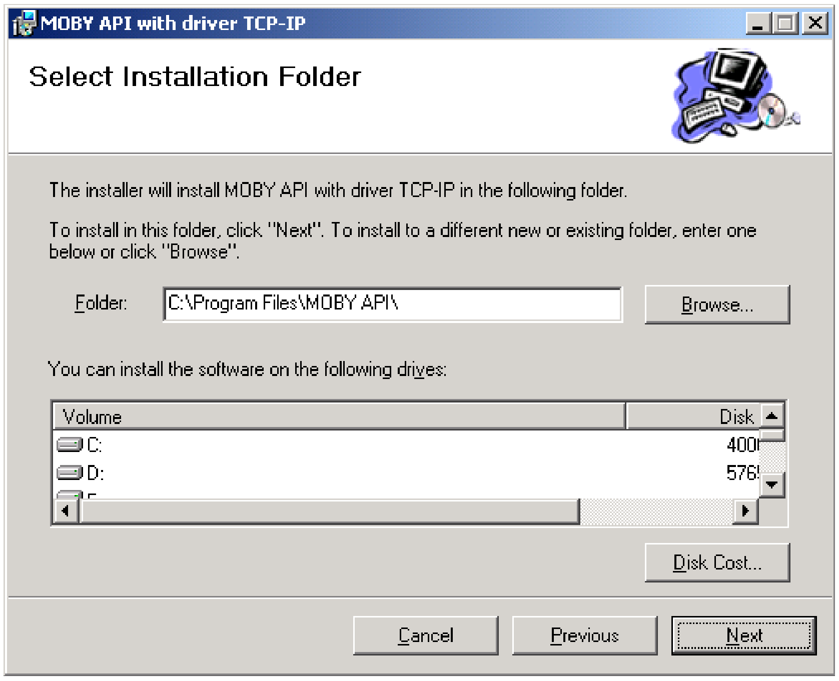
Click "Next" to continue.

WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.

Figure 2-15 Activation of the installation of the C-library (link to Ethernet)

"Next" activates the installation. The screen for selecting the directory for the MOBY API C-library (figure 2-16) appears.
"Cancel" terminates the installation (with previous confirmation).

3. Selecting the directory for the MOBY API C-library



MOBY API with driver TCP-IP

Select Installation Folder

The installer will install MOBY API with driver TCP-IP in the following folder.

To install in this folder, click "Next". To install to a different new or existing folder, enter one below or click "Browse".

Folder: C:\Program Files\MOBY API\

You can install the software on the following drives:

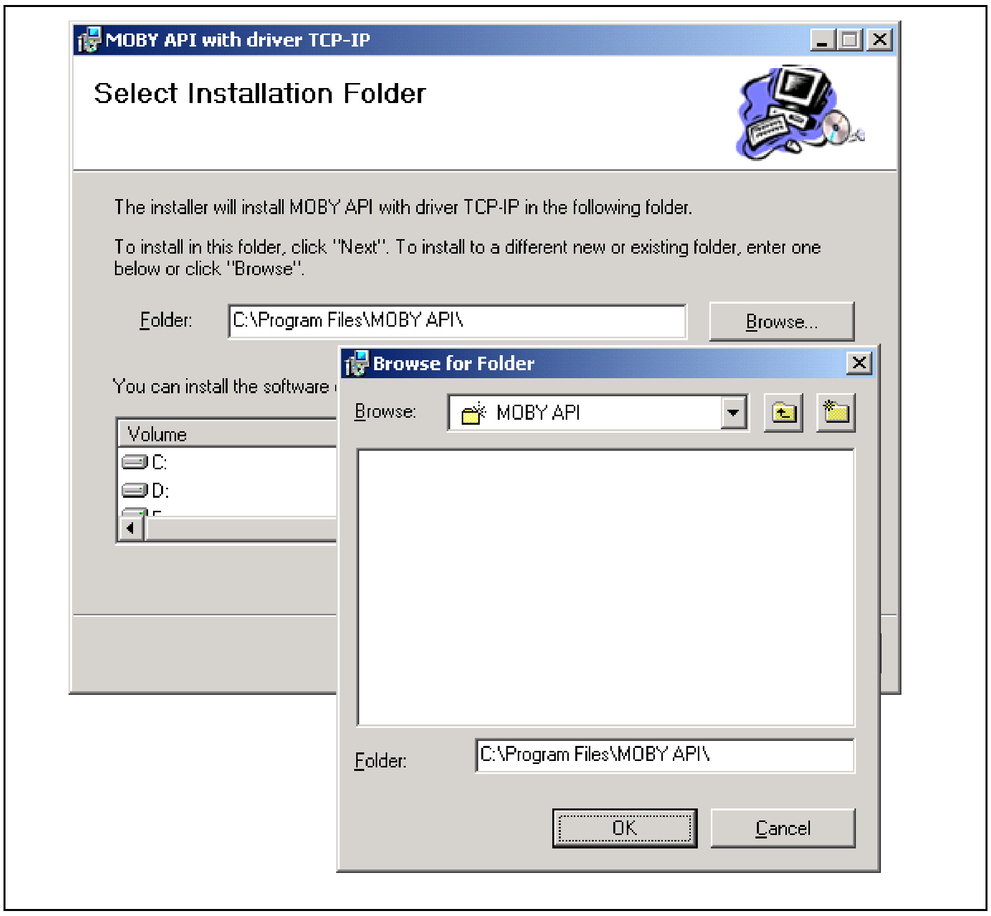
Volume	Disk
C:	400
D:	576

Figure 2-16 Selection of the directory for the installation (link to Ethernet)

The screen "Select Installation Folder" appears with the standard setting "C:\Program Files\MOBY API." This setting can be accepted with "Next." The setting can be changed in the "Folder" box or an existing directory can be selected with the Browse box (see figure 2-17).

The setting is accepted with "Next" and the screen "Confirm Installation" (figure 2-18) appears.

4. Selecting an existing directory


MOBY API with driver TCP-IP

Select Installation Folder

The installer will install MOBY API with driver TCP-IP in the following folder.

To install in this folder, click "Next". To install to a different new or existing folder, enter one below or click "Browse".

Folder: C:\Program Files\MOBY API\

You can install the software

Volume
C:
D:

Browse for Folder

Browse: MOBY API

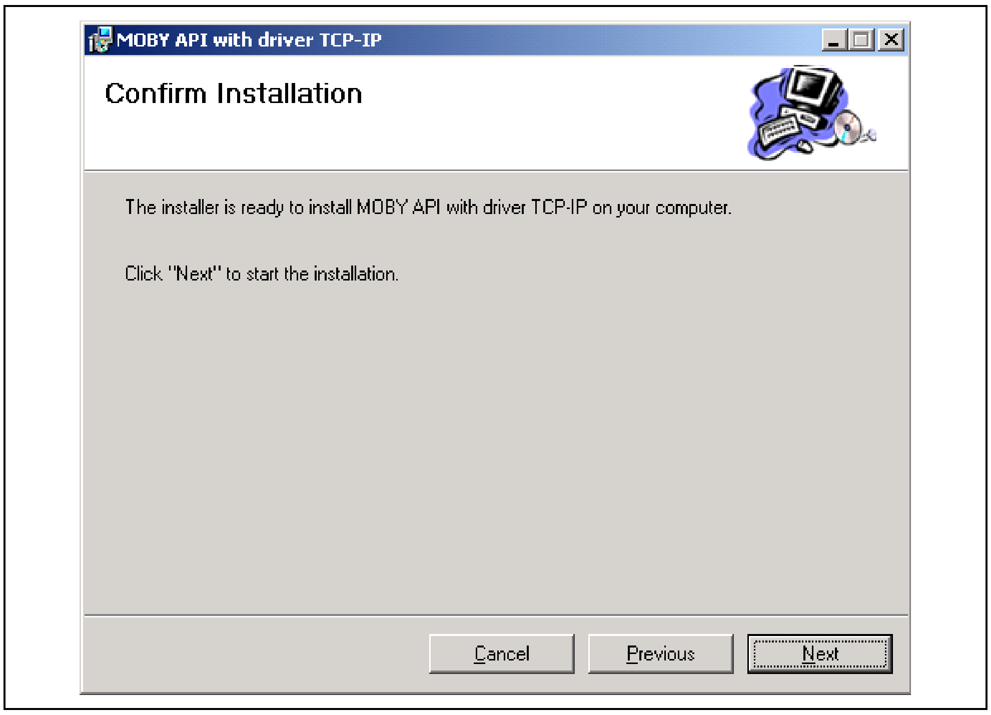
Folder: C:\Program Files\MOBY API\

Figure 2-17 Selection of an existing directory (link to Ethernet)

An existing directory can be selected in the screen "Browse for Folder," and accepted and changed in the "Folder" box.

- Press the OK button to return to the "Select Installation Folder" screen with acceptance of the directory.
- Press the Cancel button to return and cancel your selection.

5. Executing the installation of the MOBY API software


MOBY API with driver TCP-IP

Confirm Installation

The installer is ready to install MOBY API with driver TCP-IP on your computer.

Click "Next" to start the installation.

Figure 2-18 Execution of installation (link to Ethernet)

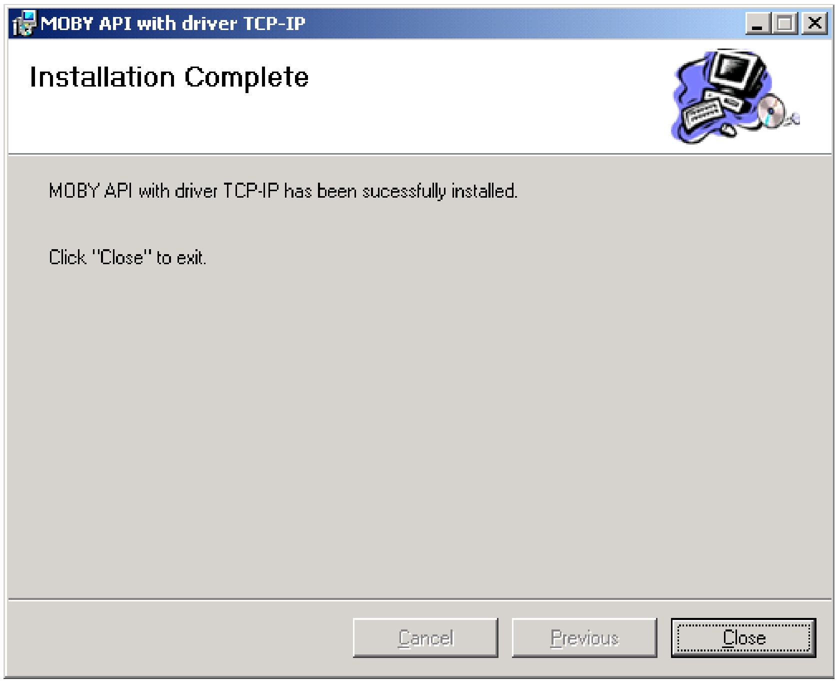
With "Next" the MOBY API software is stored for the preparation of applications under the following subdirectories within the chosen directory.

\example	EXAMPLE.ZIP	Sample applications
\include	MOBY_API_T.H	Header file for user applications
\lib	MOBY_API_T.LIB	C-library for user applications

The DLL of the dynamic link library MOBY_API_T is copied to the following directory, depending on which Windows system you are using.

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

6. Procedure for installing an existing MOBY API C-library is concluded.


MOBY API with driver TCP-IP

Installation Complete

MOBY API with driver TCP-IP has been sucessfully installed.

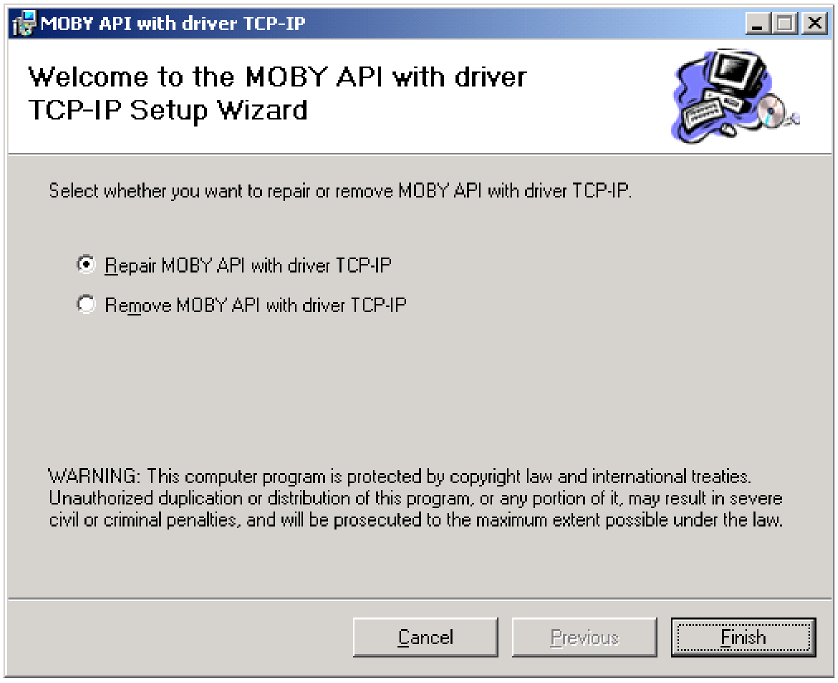
Click "Close" to exit.

Figure 2-19 Installation complete (link to Ethernet)

"Close" concludes the dialog.

Overwriting an existing C-library

7. Overwriting or de-installing an existing MOBY API C-library


MOBY API with driver TCP-IP

Welcome to the MOBY API with driver TCP-IP Setup Wizard

Select whether you want to repair or remove MOBY API with driver TCP-IP.

[✓ Repair MOBY API with driver TCP-IP]

Remove MOBY API with driver TCP-IP

WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.

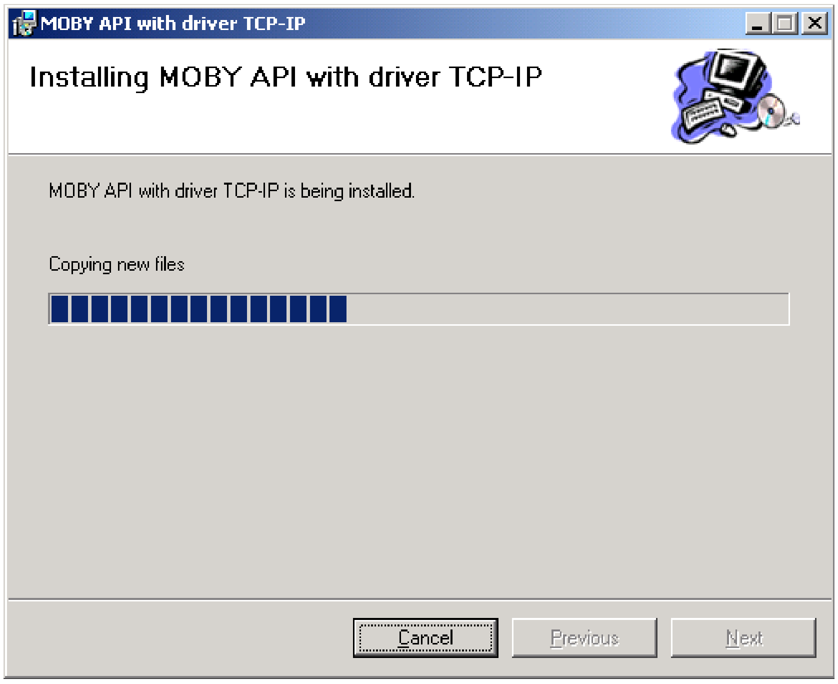
Figure 2-20 Overwriting an existing C-library (link to Ethernet)

If the existing MOBY API C-library is to be overwritten, select "Repair MOBY API with driver TCP/IP" and then start the procedure with "Finish" (see figure 2-21).

Notice

Remember to accept the new header file MOBY_API_T.H and the MOBY_API_T.LIB C-library in the development project.

8. Overwriting an existing MOBY API C-library


MOBY API with driver TCP-IP

Installing MOBY API with driver TCP-IP

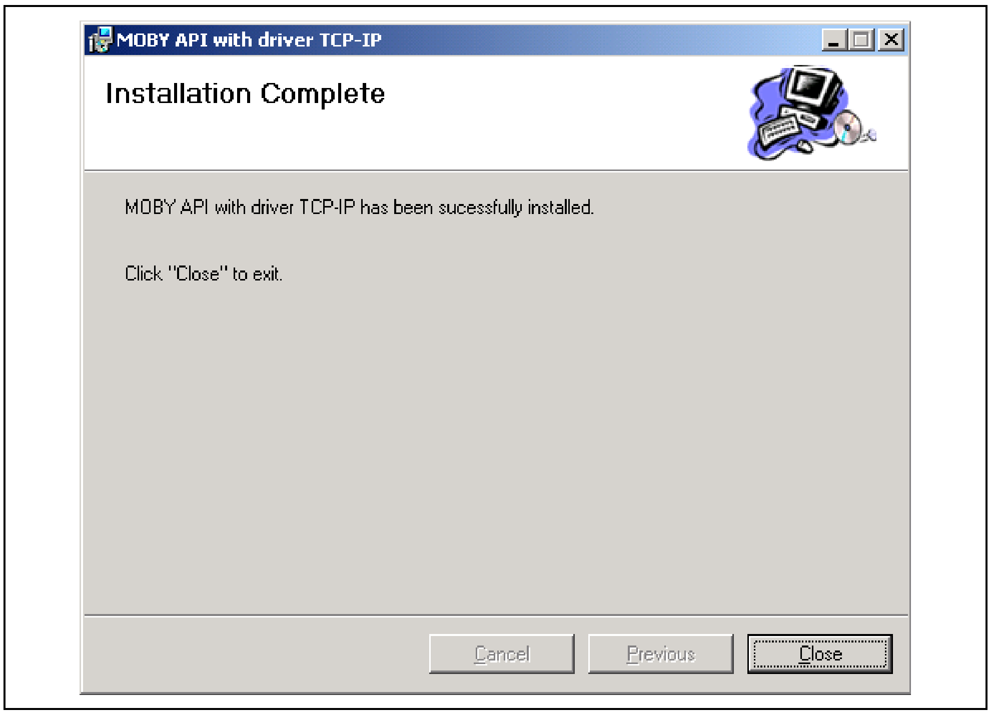
MOBY API with driver TCP-IP is being installed.

Copying new files

Figure 2-21 Screen during the "overwrite" installation procedure (link to Ethernet)

The existing MOBY API C-library is overwritten.

9. Overwriting an existing MOBY API C-library is complete (link to Ethernet).


MOBY API with driver TCP-IP

Installation Complete

MOBY API with driver TCP-IP has been sucessfully installed.

Click "Close" to exit.

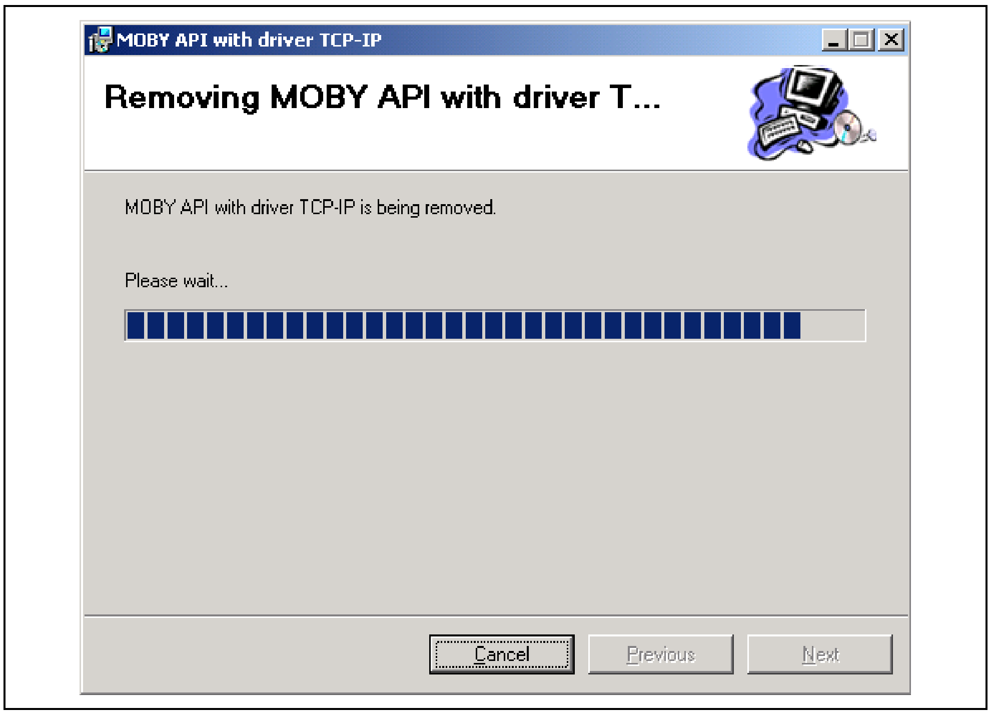
Figure 2-22 Overwrite procedure is complete.

"Close" concludes the dialog.

De-installation

When the existing MOBY API C-library is to be de-installed, select "Remove MOBY API with driver TCP/IP" (see figure 2-20) and then start the procedure with "Finish" (see figure 2-23).

10. De-installation of an existing MOBY API C-library


MOBY API with driver TCP-IP

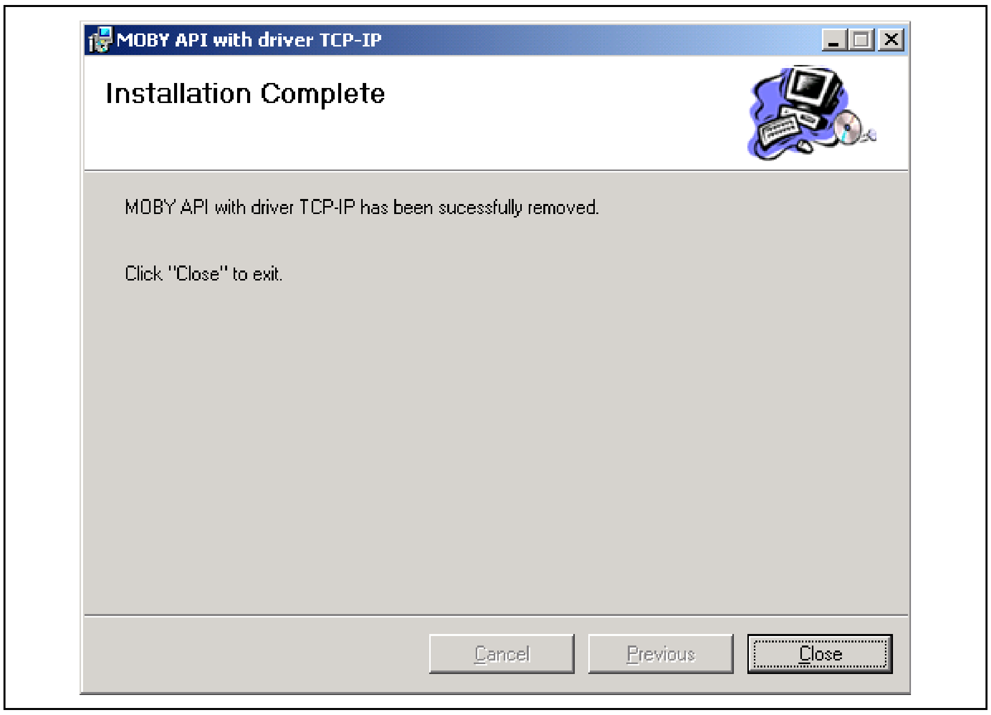
Removing MOBY API with driver T...

MOBY API with driver TCP-IP is being removed.

Please wait...

Figure 2-23 Screen during de-installation (link to Ethernet)

11. De-installation of an existing MOBY API C-library is complete.



MOBY API with driver TCP-IP has been sucessfully removed.

Click "Close" to exit.

Figure 2-24 De-installation is complete (link to Ethernet).

"Close" concludes the dialog.

2.3.1 Parameterizing ASM 480

The SLG is connected to Ethernet with the ASM 480 interface module. Communication between the application on the PC (client) and the SLG (server via the ASM 480) will only function with unique address assignment.

- The physical address (**MAC-ID** - Media Access Control Identity) is specified by the manufacturer for each ASM 480.
- In addition, each ASM 480 requires a logical address (**IP address** - Internet Protocol) with which it is addressed on the network.

The IP address may only be present once in the network. It must be parameterized on the ASM 480. In the user application on the PC, the IP address is specified in the moby_open function as a character string during the establishment of the connection.

The IP address always consists of 32 bits and is presented in decimal format (value range from 0 to 255). It is thus a character string with four number values in ASCII format, each separated by a period.

The **subnet mask** is required to determine the network. The subnet mask is similar to the IP address. It consists of four numbers separated by a period (value range from 0 to 255).

Example: IP address "157.163.170.12;" subnet mask "255.255.0.0"

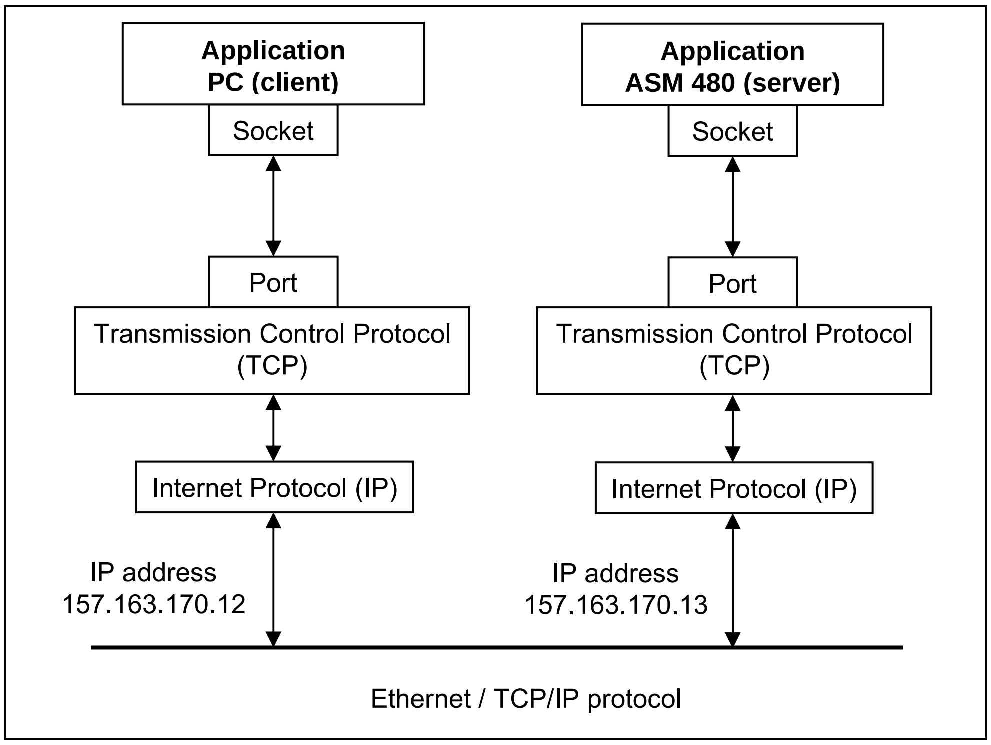

Application
PC (client)
Socket
Port
Transmission Control Protocol (TCP)
Internet Protocol (IP)
IP address 157.163.170.12
Application
ASM 480 (server)
Socket
Port
Transmission Control Protocol (TCP)
Internet Protocol (IP)
IP address 157.163.170.13
Ethernet / TCP/IP protocol

Figure 2-25 TCP/IP connection

When communication is network-overlapping between different network addresses, the **IP address of the standard gateway** must also be parameterized on the ASM 480. Without this information, IP functionality remains limited to the local subnetwork.

TCP/IP configuration: Setting the IP address, subnetwork mask and IP address of the standard gateway on the ASM 480

The TCP/IP configuration is set directly on the ASM 480 with the aid of the following.

• The four arrow keys	Up, down, right, left
• The two menu keys	ESC, OK
• The LC display	

The ASM 480 must be supplied with 24 V for the configuration.

1. Setting the IP address

a) Press OK key.	Indication: "Menu; Trace"
b) Press "down" arrow key once.	Indication: "Menu; Parameter"
c) Press OK key.	Indication: "Parameter; 3964R.1"
d) Press "down" arrow key twice.	Indication: "Parameter; IP_SETUP"
e) Press OK key.	Indication: "IP Addr Byte0:; xxx"
f) Press OK key.	Use the "up/down" arrow keys to enter the number and the "right/left" keys to enter the decimal position of the particular part of the address. Confirm your entry with the OK key.

2. Setting the subnet mask

After the IP address is set, the 4 bytes of the subnet mask (Net Mask:) are entered as described under f) for setting of the IP address.

3. Setting the standard gateway address

After the subnet mask is set, the 4-byte gateway address (Gateway:) is entered as described under f) for setting the IP address.

After all entries have been made, return to the main menu with the ESC key. Press the ESC key again to exit input mode.
The changes are stored when you press the OK key in answer to the query "Save Parameter and Restart?".

Setting the serial interface on the ASM 480

SLG U92 models can be run on the ASM 480 with the RS 232 or RS 422 interface. Depending on the SLG U92 model, the serial interface type must be parameterized on the ASM 480.

• Menu "Parameter/3964R.1":	Parameter "RTS-Control":	**RS232 RTS off** or **RS422 RTS off**

The standard setting of the ASM 480 is the interface type RS 232 (RS232 RTS OFF)

Notice

The following parameters may not be changed on the ASM 480.

• Menu "Parameter/TCP":	Parameter "CR0 Socket Type"	"Server"
• Menu "Parameter/3964R.1":	Parameter "Baudrate"	38400 (or 19200)
• Menu "Parameter/3964R.1":	Parameter "DataBits"	8
• Menu "Parameter/3964R.1":	Parameter "Stopbits"	1
• Menu "Parameter/3964R.1":	Parameter "Parity"	Odd

Other parameters for the 3964R (menu "Parameter/3964R.1":) include:

• Priority	3964R, high
• Idle time (msec)	0
• Receive mode	Byte telegram
• Send mode	Word telegram, MSB/LSB
• Error SCC	Set/clear
• Send repetition	5
• Receipt delay	150
• Character delay	50

Notice

The parameter "CR0 Port Number" is parameterized with the default value 8000 and must be specified for the moby_open function under "Parameter sPort" (see chapter 3.2.2).

• Menu "Parameter/TCP":	Parameter "CR0 Port Number"	8000

3 MOBY API C-Library

A variety of functions which can handle the entire communication are available in the C-libraries for use with the SLG or SLA on the ASM and SIM.

- MOBY_API.LIB (for serial link to PC)
- MOBY_API_T.LIB (for link to Ethernet)

C-library functions are divided into the following four function groups.

- Interface functions
- System functions
- MDS functions
- DI/DO functions

In addition, the moby_version function is available to determine the version of the dynamic link library (MOBY_API.DLL or MOBY_API_T.DLL).

The following table lists the functions and their use with SLG/ASM/SIM.

Table 3-1 Use of the library functions with SLG/ASM/SIM

Function	ASM 724 with SLA 71	ASM 824 with SLA 81	ASM 424 with ...		ASM 420 with ...		SIM 41	SLG U92	ASM 480 with SLG U92
			SLG 7x	SLG 4x	SLG 7x	SLG 4x			
Interface functions									
moby_open	x	x	x	x	x	x	x	x	x
moby_close	x	x	x	x	x	x	x	x	x
System functions									
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 preparation*

Table 3-1 Use of the library functions with SLG/ASM/SIM

Function	ASM 724 with SLA 71	ASM 824 with SLA 81	ASM 424 with ...		ASM 420 with ...		SIM 41	SLG U92	ASM 480 with SLG U92
			SLG 7x	SLG 4x	SLG 7x	SLG 4x			
MDS functions									
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
DI/DO functions									
moby_readDE	–	–	–	–	x	x	x	–	–
moby_writeDA	–	–	–	–	x	x	x	–	–
Version scan									
moby_version	x	x	x	x	x	x	x	x	x

The system, MDS and DI/DO functions generate telegrams to the SLG, ASM or SIM and always expect a response telegram from the SLG, ASM or SIM. However, they do not wait directly for the response telegram. The functions are continued after the telegram was sent successfully without errors or terminated with errors. The application must then wait for the response telegram using an initialized Windows event which must be included when the function is called. As soon as the response telegram is received, the event is set. This signals the application that the function is finished.
This procedure ensures that all functions which affect SLG, SIM or ASM do not block each other.

Notice

- With ASM 424, ASM 420 with SLG 4x and SIM 41: MDS functions with ECC offset are possible.
- The Windows event must be generated with CreateEvent.

3.1 General Information on Use of MOBY API C-Library

3.1.1 Synchronization

Every routine, which generates a response telegram from the SLG, ASM or SIM (MOBY device) to the host after the command is concluded, must be synchronized. To do this, each of these routines has a type "HANDLE" parameter. This parameter must point to an initialized Windows event which was created with CreateEvent. When one of these routines is called, the calling thread is blocked until the appropriate action is started (i.e., until a telegram is completely sent to the SLG, ASM or SIM). The calling routine then returns without waiting for a response telegram.
As soon as the response telegram is received by the SLG, ASM or SIM, the appropriate event is set to signal the calling thread that the function has now been completely concluded. Any error information in the response telegram is written to a memory word which was specified when the routine was called.

3.1.2 Matching Response Telegrams

Arriving telegrams must be matched with previous command telegrams. This is done by command numbers in the sequence of FIFO (first in, first out).

3.1.3 Unexpected Telegrams

Unexpected telegrams may sometimes be received by the SLG, ASM or SIM. A few examples of unexpected telegrams:

- Startup message “02 00 0F” hex
- Telegrams after MOBY DLL was reset which come back from SLG, ASM or SIM but whose entry was deleted from the match buffer

The calling process handles these telegrams. A callback routine can be specified which is to be called when an unexpected (i.e., unmatched) telegram arrives. Unexpected telegrams can also be ignored (see chapter 3.3.12).

3.1.4 Connection Monitoring

With an Ethernet link, the connection between the MOBY DLL and the individual SLG is monitored at the telegram level. When more than 3 seconds pass after the last telegram from the SLG, the MOBY DLL automatically sends a monitoring telegram to the SLG and expects the SLG's reply within 10 seconds. When a telegram is still not received from the SLG within these 10 seconds, the connection to the SLG is considered interrupted. The MOBY DLL closes the connection. A queued telegram is terminated with an error. When no telegrams are queued, the error does not occur until the next telegram. Before communication can be resumed, the application must start the connection again with moby_open and moby_start.

3.1.5 Errors

Telegrams from the SLG, ASM or SIM may sometimes have errors or not be completely received. "Moby_stop" is then called implicitly (i.e., use of the interface is blocked and all waiting jobs are terminated with an error message). Before the interface can be used again, "moby_start" must be executed (i.e., a RESET of the SLG, ASM or SIM).

Notice

All potentially affected interfaces are blocked when an error cannot be definitely related to an open interface (e.g., hardware error on an ASM 824/724/424 channel).

Notice

If Windows exceeds the monitoring time during the sending procedure, an unexpected telegram may occur depending on where the telegram was interrupted. The callback routine will report this telegram if you have called the moby_unexpect function. Ignore the telegram.

3.1.6 Opening, Using and Closing Interfaces

Before an SLG, ASM or SIM (MOBY device) can be addressed with MOBY API, the appropriate interface must be opened and initialized. This is done with the "moby_open" and "moby_start" commands. The interface can then be accessed with read and write commands. Before the end of the program, the programmer must deactivate the interface with "moby_stop" and then close it with "moby_close." This procedure is illustrated graphically in figure 3-1.

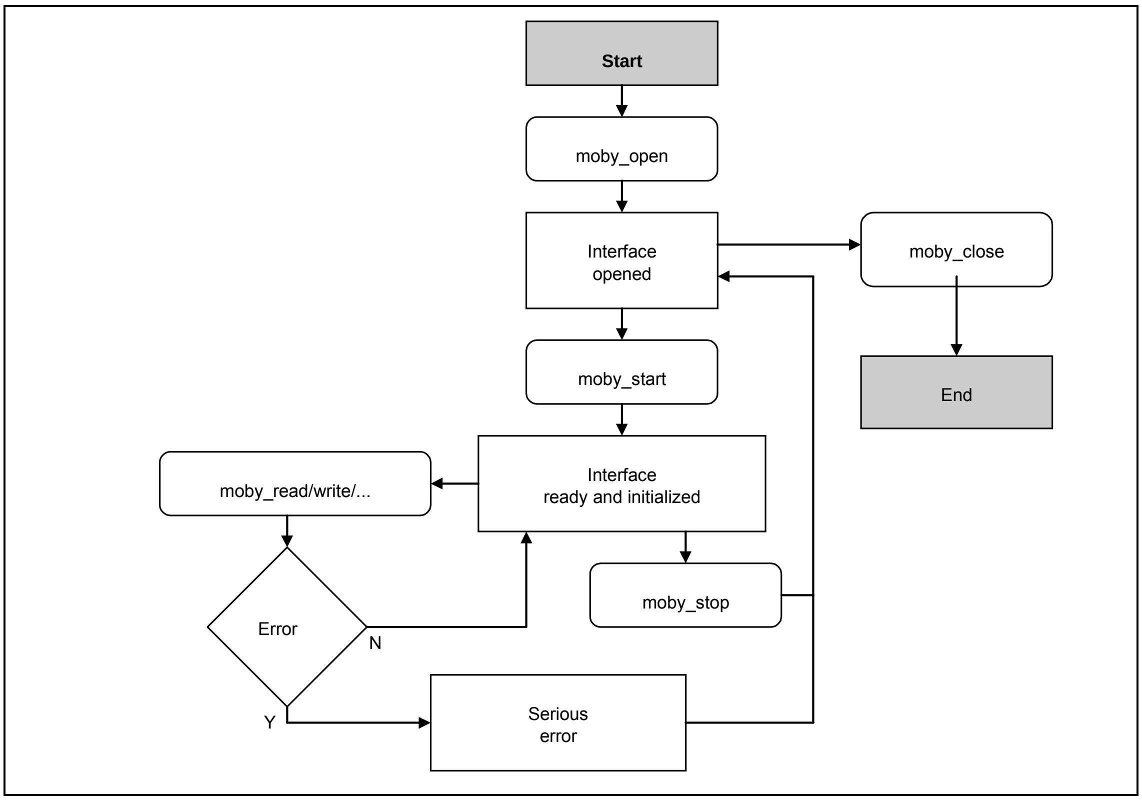


Figure 3-1 Function sequence when MOBY API is used

Figure 3-1 also shows what happens when a serious communication error or an error of line monitoring occurs. The "moby_stop" command is automatically given which returns the interface to a defined but inactive state.

Notice

When an application is terminated externally (e.g., with CRTL-ALT-DEL) and the application is used by the MOBY API C-library and has at least one channel open, it may sometimes happen under Windows 95/98 that system resources are not enabled correctly. This may cause problems when MOBY API programs are executed later. If this happens, start the system again.

3.1.7 Use of Several Communication Channels

The MOBY API C-library permits the use of several communication channels on one physical interface. This can be done by repeated use of the "moby_open" command. Make sure that all communication channels of a physical interface are to be opened by the same process (i.e., by the same application).

3.1.8 ERRx.TXT Log File(s)

When the application with TCP/IP is used, the system generates an ERRx.TXT log file with a maximum length of 7 kbytes for each connection (i.e., communication channel). x stands for the channel number.
The log file(s) is/are stored under the path from which the application was called. The log file contains internal system diagnostic data.

3.2 Interface Functions

The interface functions moby_open and moby_close open and close the communication channel for communication via:

- the serial interface or
- the Ethernet interface

At this time, the Ethernet interface can only be used for MOBY U with the SLG U92 as MOBY type "MOBY_Uc" (see chapter 3.3.1) with MOBY U commands for single-tag or multitag operation.

3.2.1 Function - moby_open for the Serial Interface

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

This function opens the specified MOBY device with the help of the driver. If the command is concluded successfully, the "handle" parameter is used to return a handle to the opened MOBY device. This handle must be included with all subsequent calls which access this device. Do not forget that an opened MOBY device must be parameterized and initialized beforehand with "moby_start."

Parameter	Type	Description
comStr	char *	Character string with the name of the serial interface COM1, COM2, COM3 or COM4
channel	int	Channel number under which the SLG, SIM, or SLA is addressed. 1 to 4 for SLA or SLG 4x on ASM 424 or 0 if SLG U92, SLG xx on ASM 420 or SIM 4x.
handle	mobyHandle_t *	Pointer to the handle of the serial interface and the channel number (response value after successful function). The returned handle is used for all other functions as the input parameter to identify the serial interface and the channel number.

Return value:

≥ 0	No error, command executed.
< 0	Windows is unable to open the serial interface (see chapter 3.7.1).

Notice

The channel number is automatically added for each function which is related to this interface.

3.2.2 Function - moby_open for the Ethernet Interface

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

This function opens the specified MOBY device with the aid of the driver. The "handle" parameter is used to return a handle to the opened MOBY device when a command is successfully concluded. This handle must be transferred with all subsequent calls which access this device. An open MOBY device must be parameterized before with "moby_start" and initialized, however.

Parameter	Type	Description
comStr	char *	IP address of the ASM 480 as character string (see chapter 2.3.1) Example: “157.163.170.12”
channel	int	Channel number under which the SLG is addressed 0 for SLG U92
handle	mobyHandle_t *	Pointer to the handle of the Ethernet interface (return value after erroneous function execution). The returned handle is used for all other functions as the input parameter for identification of the Ethernet interface.
sPort	unsigned short	Port number under which the ASM 480 is addressed (see chapter 2.3.1).

Return value:

≥ 0	No error, command executed.
< 0	Windows is unable to open the Ethernet interface (see chapter 3.7.1).

3.2.3 Function - moby_close

mobyErr_t moby_close (mobyHandle_t handle);

This function closes the specified MOBY device with the help of the driver. The MOBY device can then no longer be accessed. This routine is required for communication with the MOBY device with "moby_start" (see chapter 3.3.1). However, it is not used until after the user calls "moby-stop" (see chapter 3.3.2). Otherwise, an error is returned.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number or the Ethernet interface

Return value:

≥ 0	No error, command executed.
< 0	Windows is unable to close the serial interface or the Ethernet interface (see chapter 3.7.1).

3.3 System Functions

The system functions are used to control communication with an SLG or SLA on an ASM or a SIM. They provide the function framework for the MDS functions (see chapter 3.4) but do not affect the MDS directly.

Table 3-2 Assignment of the system functions to the interfaces

Function	Serial Interface	Ethernet Interface
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 Function - moby_start

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

This function parameterizes and initializes an already opened MOBY device. Among others, a RESET telegram (with or without parameter transfer) is also sent to the SLG, ASM or SIM. Not until this telegram is concluded (successfully or not) is the interface ready for other telegrams.
When this function is to be called again after successful execution, the moby_stop function must be executed first.
The parameters for the RESET telegrams are transferred with a union-data structure (see chapter 4.1 for MOBY_API.H header file or chapter 4.3 for MOBY_API_T.H header file). Evaluation of the union-data structure is based on a constant with which the MOBY type (system) is specified.

Table 3-3 Constants for the MOBY type

Constant (See Type Parameter)	MOBY Type	Interface		Commentary
		Serial	Ethernet	
"MOBY_E"	SLG 7x on ASM 420 or SLA 7x on ASM 724	X	–	
"MOBY_F"	SLA 8x on ASM 824	X	–	
"MOBY_I" or "MOBY_Ia"	SIM 41, SLG 4x on ASM 420 or SLG 4x on ASM 424	X	–	When the option with parameter is used, the RESET telegram is always used to set the interval parameters.
"MOBY_Ib"	SIM 41, SLG 4x on ASM 420 or SLG 4x on ASM 424	X	–	When "MOBY_Ib" is specified, the second parameter version - use of extra parameters (e.g., LED Settings) - is used.
"MOBY_Ua"	SLG U92	X	–	Short RESET telegram (MOBY I call-compatible, no multitagging)
"MOBY_Ub"	SLG U92	X	–	Long RESET telegram (MOBY I call-compatible, no multitagging)
"MOBY Uc"	SLG U92	X	–	Long RESET telegram (MOBY U commands for single-tagging or multitagging operation) • Value in param = 1 (single-tagging) • Value in param > 1 (multitagging)
	SLG U92 on ASM 480	–	X	

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number or the Ethernet interface
type	mobyTypeee_t	Character string with the name of the MOBY system "MOBY_E" "MOBY_F" "MOBY_I" "MOBY_Ia" "MOBY_Ib" “MOBY_Ua” “MOBY_Ub” “MOBY_Uc”

eccON	BOOL	Boolean value for MDS functions with or without ECC offset
		FALSE = No ECC offset
		TRUE = ECC offset

param	mobyParameters_t *	Pointer for parameter transfer NULL = No parameter transfer (standard RESET) or pointer to the buffer in which the parameters to be transferred are located (RESET parameters)
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Windows event due to which the application is waiting for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte of the response telegram with any error codes and interface errors is to be stored (see chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

For RESET parameters, see the documentation of your MOBY.

• SLG U92	See appendices B.2.1.2.1, B.3.1.2.1 and B.4.1.2.1.
• SLG U92 on ASM 480	See appendix B.4.1.2.1.
• SLG on ASM 420	See /01/
• SIM 41	See /02/
• SLA 8x on ASM 824	See appendix A.6.2
• SLA 7x on ASM 724	See appendix A.6.2
• SLG 4x on ASM 424	See appendix A.6.2

With MOBY E and I, the moby_start function corresponds to the RESET telegram with or without parameter transfer.

3.3.2 Function - moby_stop

mobyErr_t moby_stop (mobyHandle_t handle);

This function resets communication with an SLG or SLA on an ASM or SIM and the MOBY DLL. After its execution, no further commands can be sent to the device until another "moby_start." Still queued commands are terminated and the waiting threads are informed accordingly.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number or the Ethernet interface

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

This routine does not send telegrams to the SLG, ASM or the SIM.

3.3.3 Function - moby_next

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

If the last processed MDS is still in the field, you can already activate processing of the next MDS after this function.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Windows event due to which the application is waiting for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte of the response telegram with any error codes and interface errors is to be stored (see chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

3.3.4 Function - moby_end

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

This function is used to deactivate the standby time (parameterized in RESET telegram) of the MDS which was last processed and is still in the field of the SLG. This is done to reduce the current consumption of the MDS.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number
mode	unsigned char	00 hex = Processing of the MDS is finished and the parameterized standby time is to be deactivated. No more communication with this MDS is to take place. The MDS will exit the field of the SLG. The SLG removes the MDS from the processing list but retains the MDS in the presence list until the MDS leaves the field of the SLG. 01 hex = There is a pause in MDS processing and the parameterized standby time is to be deactivated. The MDS has not yet departed the field of the SLG. Communication with the MDS will take place at least one more time. The SLG keeps the MDS in both the processing list and the presence list.
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Window event which the application uses to wait for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte is to be stored with possible error code and interface error. See chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

The END command may only be sent to the SLG after the command moby_write, moby_read, moby_init or moby_status and no command may be queued on the SLG. The antenna must be on. If not an error message occurs.
If the MDS has already left the field of the SLG and the mode is 01, an error message is generated. If, however, another MDS has entered the field of the SLG and the mode is 01, an error message is also generated.

Notice

The command can only be used with MOBY U (type MOBY_Ua or MOBY_Ub – see table 3-3). This is MOBY U without multitagging.

3.3.5 Function - moby_s_end

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

This function is used to "specifically" or "non-specifically" ignore the standby time (parameterized in the RESET telegram) of an MDS that is in the antenna field of the SLG U92.

- A "specific" read call means that the call is sent with the ID number of the MDS. More than one MDS may be located in the antenna field. You can determine the ID number of the MDS with the GET function.
- A "non-specific" read call means that the call is sent without the ID number of the MDS. Only one MDS may be located in the antenna field.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number or the Ethernet interface
idData	uInt *	Value of 20 to 232 - 1 ID number of MDS 0 Only one MDS is located in the field and a "non-specific" read call is to be used.
mode	unsigned char	00 hex = Processing of the MDS is finished and the parameterized standby time should be ignored. No more communication is to take place with this MDS. The MDS will exit the field of the SLG. The SLG removes the MDS from the processing list but keeps the MDS in the presence list until the MDS has actually exited the field of the SLG. 01 hex = There is an MDS processing pause and the parameterized standby time is to be ignored. The MDS hasn't left the field of the SLG yet. At least one further communication with the MDS is to take place. The SLG continues to keep the MDS in the processing list and in the presence list.
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Window event which the application uses to wait for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte is to be stored with possible error code and interface error. See chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

The moby_s_end command may only be sent to the SLG after the command moby_s_write, moby_s_read, moby_s_init or moby_s_statusMDS and no command may be queued yet on the SLG. The antenna must be on. If not, an error message is generated.
If the MDS that is "specifically" called with the specified ID number is not in zone 1, the command is terminated with an error. If the MDS has already left the field of the SLG and mode = 0 was selected, an error message is generated. If, however, another MDS has entered the field of the SLG and mode = 01 was selected, an error message is also generated. If more than one MDS is located in zone 1 and a "non-specific" call occurs, the command is terminated with an error.

3.3.6 Function - moby_setANT

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

This function is used to turn the antenna of the read/write device (SLG) on or off.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number or the Ethernet interface
mode	unsigned char	01 hex = Turn on antenna 02 hex = Turn off antenna
sync	HANDLE	Handle to synchronization with the response telegram. Initialized Windows event which the application uses to wait for the response telegram.
err	mobyErr_t *	Pointer to a structure in which the status byte and any error code and interface errors are to be stored. See chapter 3.7.2.

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

The moby_setANT command may only be sent to the SLG when no command is still queued on the SLG.
An MDS may already be present in the field of the SLG at the time the antenna is turned on.
When an MDS is located in the field of the SLG when the antenna is turned off, it is reported as "not present" if the presence check is being used.

3.3.7 Function - moby_repeat [1]

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

This function is used to repeat the command which was executed last.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number or the Ethernet interface
mode	unsigned char	00 hex = Repeat last command until moby_repeat is finished with mode = 01. 01 hex = Conclude repetition. A started command will be executed until the end.
sync	HANDLE	Handle to synchronization with the response telegram. Initialized Windows event which the application uses to wait for the response telegram.
err	mobyErr_t *	Pointer to a structure in which the status byte and any error code and interface errors are to be stored. See chapter 3.7.2.

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

The command with mode = 0 may only be sent when a command is queued for execution (e.g., moby_read).

1 *In preparation.*

3.3.8 Function - moby_status

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

This function is used to query the status of an SLG or SLA on an ASM or SIM.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number
status	mobyStatus_t *	Pointer to the structure in which the read status information is to be stored
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Windows event due to which the application is waiting for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte of the response telegram with any error codes and interface errors is to be stored (see chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

Only one waiting status-telegram is permitted. Status queries which are sent when a status query is still waiting to be answered are rejected with an error message. Status information varies depending on the particular MOBY. The data do not become valid until the synchronization handle signals receipt of the response telegram. For content and meaning, see documentation /01/ to /05/ of your MOBY or also appendix A.6.6 for ASM 424/724/824.

3.3.9 Function - moby_statusU

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

This function is used to poll the status of an SLG U92.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number or the Ethernet interface
status	mobyStatusU_t *	Pointer to the structure in which the status and diagnostic information of the response telegram is to be stored.
sync	HANDLE	Handle to synchronization with the response telegram. Initialized Windows event which the application uses to wait for the response telegram.
err	mobyErr_t *	Pointer to a structure in which the status byte of the response telegram and any error code and interface errors are to be stored. See chapter 3.7.2.

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

Only one queued status telegram is permitted at a time. All status queries which are issued at a time when a status query is still unanswered will be rejected as an error.
The status information is specific to MOBY. The data are not valid until the synchronization handle signals the receipt of the response telegram. For content and meaning, see appendix B.2.2.2.2.

3.3.10 Function - moby_anw

mobyErr_t moby_anw (mobyHandle_t handle, moby_AnwCallback_t cbroutine);

This function is used to report the presence of the MDS in the field of the SLG. The presence message is sent asynchronously by the SLG and is called as a callback routine.

When "operation with presence check" was set in the RESET telegram, the SLG sends a telegram with the number of MDSs in the field after each change in presence in the SLG's field. If one MDS leaves the field and another MDS enters the field at the same time, two telegrams are sent. If several MDSs enter the field simultaneously, each MDS creates a presence message. The same applies when MDSs leave the field. The transfer value and the number of MDSs in the field are defined with "typedef void (CALLBACK *moby_AnwCallback_t)(unsigned char anwstatus);" in the header file MOBY_API.H or MOBY_API_T.H. See also chapter 4.1 or chapter 4.3.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number or the Ethernet interface
cbroutine	Moby_AnwCallback_t	Callback routine for the presence message from the SLG. MDS has arrived or MDS has departed.

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

If the moby_anw function does not exist in the user program and "operation with presence check" is set in the RESET telegram, each presence message is handled as an unexpected telegram. See chapter 3.3.12 (moby_unexpect function).

3.3.11 Function - moby_diagnose

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

This function is used to read the diagnostic data from the SLG.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number or the Ethernet interface
mode	unsigned char	02 hex = Request last n function calls 03 hex = Request last n error messages 04 hex = Request last n identified MDSs
diagnose	mobyDiagnose_t *	Pointer to the structure in which the read diagnostic information is to be stored
sync	HANDLE	Handle to synchronization with the response telegram. Initialized Windows event which the application uses to wait for the response telegram.
err	mobyErr_t *	Pointer to a structure in which the status byte and any error code and interface errors are to be stored. See chapter 3.7.2.

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

Only one unanswered diagnostic telegram is permitted at a time. Any diagnostic query which is sent when an unanswered diagnostic query still exists is rejected with an appropriate error.
The moby_diagnose command may be sent to the SLG at any time. It is executed immediately. When an MDS command such as moby_write, moby_read or moby_init is queued on the SLG, it is retained.
The diagnostic information is MOBY U-specific. The data do not become valid until the synchronization handle signals the receipt of the response telegram. For content and meaning, see appendices B.2.2.2.3 to B.2.2.2.5.

3.3.12 Function - moby_unexpect

**mobyErr_t moby_unexpect (mobyHandle_t handle,
moby_UnexpCallback_t cbroutine);**

This function is used to specify how unexpected messages are to be handled. If the parameter "cbroutine" is NULL, all unexpected messages are ignored after this routine is called. If not NULL, this parameter must be the address of a valid callback routine. This routine is jumped to when unexpected messages arrive so that the user program can react to this event. When the callback routine is triggered, the first 3 bytes of the unexpected telegram and the actual telegram length are transferred. These transfer values are defined in the MOBY_API.H or MOBY_API_T.H header file as "typdef void (CALLBACK *moby_UnexpCALLback_t) (unsigned char ch1, unsigned char ch2, unsigned char ch3, int laenge);" (see chapter 4.1 or chapter 4.3).

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number or the Ethernet interface
cbroutine	moby_UnexpCallback_t	Callback routine for unexpected telegrams from SLG, ASM or SIM

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

Unexpected telegrams which can clearly be interpreted as error messages (e.g., occurrence of an invalid telegram) are evaluated as errors and handled as described in chapter 3.1.4.

3.4 MDS Functions

MOBY API offers MDS functions in two versions.

- "Non-specific" MDS functions
- "Specific" MDS functions

The "non-specific" MDS functions are listed below.

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

These functions can be used to read data from any MDS or write data to any MDS. Since you cannot call a specific MDS, only one MDS may be located in the field of the antenna at the time the function is called and executed. Multitagging is not possible.

These functions are executed based on the eccON parameter (with or without ECC offset) specified in the moby_start function. See chapter 3.3.1).

See documentation /01/ to /06/ or also appendix A.8 for ASM 424/724/824 for which address areas on the MDS can be read and written by the individual MOBY systems.

See table 3-1 and chapters 3.4.1 to 3.4.6 for which functions can be used with which parameterizations with the MOBY systems.

Notice

The “non-specific” MDS functions can only be used via the serial interface (not via the Ethernet interface).

The "specific" MDS functions are listed below.

- moby_s_read
- moby_s_getID
- moby_s_write
- moby_s_init
- moby_s_copy
- moby_s_statusMDS
- moby_s_readOTP
- moby_s_writeOTP

These functions can be used to read data from a certain MDS or write data to a certain MDS. They can select a certain MDS by its MDS identification number (serial number). If you don't specify an MDS number for these functions (except for moby_s_getID and moby_s_copy), they behave as "non-specific" function calls.

The moby_s_getID function determines the identification number/numbers of the MDS/MDSs located in the antenna field. You can process several MDSs in the antenna field. Multitagging is possible. In addition, the moby_s_copy function lets you copy data from one MDS (source) to another MDS (destination) without having to use the application.

The "specific" MDS functions (i.e., multitag mode is possible) can only be used with MOBY U. With the moby_start function the parameter "type" must be set to MOBY_Uc (MOBY U with commands with single-tag and multitag capability). See table 3-1 and chapters 3.4.7 to 3.4.15 for which functions can be used with which parameterization with MOBY U.

Table 3-4 Assignment of the MDS functions to the interfaces

Function	Serial Interface	Ethernet Interface
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 Function - moby_read

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

The moby_read function is used to read data from the MDS located in the antenna field of the SLG, SLA or SIM. It is an "unspecific" read call since the MDS cannot be identified by an ID number.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number
mdsAddress	uInt	Start address of the data to be read from the MDS ≥ 0 0 to maximum length of the user data minus 1. The address plus the data length must be less than the end address on the MDS.
data	unsigned char *	Pointer to the buffer in which the read data are to be stored
length	uInt	Number of bytes to be read > 0 1 to maximum of 248 bytes
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Windows event due to which the application is waiting for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte of the response telegram with any error codes and interface errors is to be stored (see chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

The read data are stored in the buffer specified by the "data" and "length" parameters. These data do not become valid until the synchronization handle signals that the response telegram has arrived with the read data.
With ASM 724/ASM 424, only a maximum of 234 bytes can be read with one telegram. With ASM 824, up to 192 bytes can be read with one telegram.

3.4.2 Function - moby_getID

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

The moby_getID function is used to read the identification number (serial number) of the MDS. This is only possible with the MDSs of MOBY E, MOBY F and MOBY U.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number
idData	unsigned char *	Pointer to the buffer in which the read ID number is to be stored
idLength	uInt	Length of the ID number of the MDS in bytes: > 0 Length of the ID number: • MOBY E: ID length = 4 bytes • MOBY F, MDS F1xx: ID length = 5 bytes • MOBY F, MDS F4xx: ID length = 4 bytes • MOBY U: ID length = 4 bytes
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Windows event due to which the application is waiting for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte of the response telegram with any error codes and interface errors is to be stored (see chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

The start address of the ID on the MDS is handled internally.

• MOBY E:	Start address = 1FF0 Hex
• MOBY F, MDS F1xx:	Start address = 0000 Hex
• MOBY F, MDS F4xx:	Start address = 0000 Hex

The read data are stored in the buffer specified by the parameters "idData" and "idLength." These data do not become valid until the synchronization handle signals that the response telegram has arrived with the read data.

3.4.3 Function - moby_write

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

The moby_write function is used to write the data to the MDS located in the antenna field of the SLG, SLA or SIM. It is an unspecific write access.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number
mdsAddress	uInt	Start address on the MDS for the data to be written. ≥ 0 0 up to maximum length of the user data minus 1. Address plus data length must be less than the end address on the MDS.
data	unsigned char *	Pointer to the buffer in which the data to be written are stored
length	uInt	Number of bytes to be written > 0 1 to maximum of 248 bytes
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Windows event due to which the application is waiting for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte of the response telegram with any error codes and interface errors is to be stored (see chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

With ASM 724/ASM 424, only a maximum of 234 bytes can be written with one telegram. With ASM 824, a maximum of 192 bytes can be written with one telegram.

3.4.4 Function - moby_init

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

The moby_init function is used to initialize (write the entire MDS with one byte) the MDS located in the antenna field of the SLG, SLA or SIM. It is an unspecific initialization call.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number
setVal	unsigned char	1 byte in hex format with which the entire MDS is to be written 00 hex to FF hex
mdsLength	uInt	Number of bytes to be deleted on the MDS. For the number of bytes to be deleted (length of the user data), see the technical description.
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Windows event due to which the application is waiting for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte of the response telegram with any error codes and interface errors is to be stored (see chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

When the number of bytes to be deleted is less than the length of the user data of the MDS, the memory of the MDS is written with the specified byte from address 0 to address (length - 1). The rest of the memory is not changed.

3.4.5 Function - 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);

This function is used to request status and diagnostic data of an MDS.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number
statusMDS	mobyStatusMDS_t *	Pointer to the structure in which the status and diagnostic information of the response telegram are to be stored
mode	unsigned char	00 hex = Request status and diagnostic data of an MDS
cweek	unsigned char	01 to 35 hex = Current calendar week (1 to 53) for determination of remaining battery life FF hex = No determination of remaining battery life
year	unsigned char	01 to 63 hex = The last two positions of the current year (starting with 01) for determination of remaining battery life FF hex = No determination of remaining battery life
sync	HANDLE	Handle to synchronization with the response telegram. Initialized Windows event which the application uses to wait for the response telegram.
err	mobyErr_t *	Pointer to a structure in which the status byte and any error code and interface errors are to be stored. See chapter 3.7.2.

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

The function sequence is dependent on the fields cweek and year.

a) If the value in the two fields is within the value range, the remaining battery life is output in the response telegram.

b) If the two fields contain FF hex, the remaining battery life cannot be calculated. It is specified as FFFF hex days. The status data of the MDS are output anyway.

After the call, the buffer contains the status and diagnostic data of the MDS. For the format, see the MOBY U documentation /06/.

3.4.6 Function - moby_readOTP

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

This function can be used to read the OTP (One Time Programmable) memory with a block of 16 bytes (= 128 bits).

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number
data	unsigned char *	Pointer to the buffer in which the data read from the OTP memory are to be stored The buffer must have a length of at least 16 bytes.
sync	HANDLE	Handle to synchronization with the response telegram. Initialized Windows event which the application uses to wait for the response telegram.
err	mobyErr_t *	Pointer to a structure in which the status byte of the response telegram and any error code and interface errors are to be stored. See chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

The OTP memory has a length of 16 bytes (= 128 bits) and can only be read once as a whole. This means that the moby_read OTP function always reads 16 bytes. The buffer for the read data must have a minimum length of 16 bytes. Otherwise the memory area is overwritten.

3.4.7 Function - moby_writeOTP

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

This function is used to write the OTP (One Time Programmable) memory once with a block of 16 bytes (= 128 bits).

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number
data	unsigned char *	Pointer to the buffer in which the data to be written to the OTP memory are stored. The buffer must have a length of at least 16 bytes.
sync	HANDLE	Handle to synchronization with the response telegram. Initialized Windows event which the application uses to wait for the response telegram.
err	mobyErr_t *	Pointer to a structure in which the status byte of the response telegram and any error code and interface errors are to be stored. See chapter 3.7.2.

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

The OTP memory has a length of 16 bytes (= 128 bits) and can only be written once as a whole. This means that the moby_write OTP function always writes 16 bytes.

3.4.8 Function - 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);

You can use this function to "specifically" or "non-specifically" read data from an MDS which is located in the antenna field of the SLG U92.

- With a "specific" read call the call contains the identification number of the MDS. More than one MDS may be located in the antenna field. The identification number of the MDS can be determined with the GET function.
- With a "non-specific" read call the call does not contain the identification number of the MDS. Only one MDS may be located in the antenna field.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number or the Ethernet interface
idData	uInt *	Value of 20 to 232 - 1 ID number of MDS 0 Only one MDS is located in the field and a "non-specific" read call is to be used.
mdsAddress	uInt	Start address of the data on the MDS to be read ≥ 0 0 to maximum length of user data minus 1 The address plus data length must be less than the end address on the MDS.
data	unsigned char *	Pointer to the buffer in which the read data are to be stored
length	uInt	Number of bytes to be read > 0 1 to maximum of 244 bytes
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Windows event due to which the application is waiting for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte of the response telegram with any error codes and interface errors is to be stored (see chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

The MOBY_s_read command may only be sent to the SLG U92 when no command is queued yet on the SLG U92. The antenna must be on. Otherwise an error message is generated.
The command is terminated with an error if the MDS with the specified identification number that is "specifically" called is not located in zone 1.
If no MDS is located in zone 1 and a "non-specific" call occurs, the command waits until an MDS moves into zone 1 or the RESET command arrives.
If more than one MDS is located in zone 1 when a "non-specific" call arrives, the command is terminated with an error.

Notice

The data that were read are stored in the buffer specified by the "data" and "length" parameters. These data do not become valid until the synchronization handle signals that the response telegram has arrived with the read data.

Notice

With SLG U92 with ASM 480 (TCP/IP), a maximum of only 228 bytes can be read with one telegram.

3.4.9 Function - moby_s_getID

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

You can use this function to query which MDSs are located in the field. At the same time you can read data from these MDSs.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number or the Ethernet interface
getInfo	mobyMtget_t *	Pointer to the structure in which the number of polled MDSs, the ID numbers of the MDSs and, if necessary, the read MDS data are to be stored.
address	int	Start address of the data on the MDS to be read ≥ 0 0 to maximum length of the user data minus 1 The address plus data length must be less than the end address on the MDS. - 16 = FFF0 hex Start address of the OTP memory
length	uInt	Number of bytes to be read ≥ 0 0 to maximum length x x = (242 – (4 * bunch size)) / bunch size 16 Length of the OTP memory
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Windows event due to which the application is waiting for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte of the response telegram with any error codes and interface errors is to be stored (see chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

The moby_s_getID command may only be sent to the SLG U92 when no command is yet queued on the SLG U92. The antenna must be on. Otherwise an error message is generated.

Notice

The 128 bits of user information in the OTP memory are addressed with the start address -16 (FFF0 hex). The 128 bits of user information are written to the MDS with the WRITE command. All 128 bits of information must be requested with the GET call.

Notice

With the SLG U92 with ASM 480 (TCP/IP), a maximum of only 226 bytes (MDS ID numbers and user data) can be read with one telegram. This means:
parameter length = x = (226 – (4 * bunch size)) / bunch size.

3.4.10 Function - moby_s_write

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

You can use this function to "specifically" or "non-specifically" write data to an MDS which is located in the antenna field of the SLG U92.

- A "specific" call to write contains the identification number of the MDS. More than one MDS may be located in the antenna field. You can determine the ID number of the MDS with the GET function.
- A "non-specific" call to write does not contain the identification number of the MDS. Only one MDS may be located in the antenna field.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number or the Ethernet interface
idData	uInt *	Value from 20 to 232 - 1 Identification number of the MDS 0 When there is only one MDS in the field and a "non-specific" write call is to be made
mdsAddress	uInt	Start address on the MDS for the data to be written ≥ 0 0 to maximum length of the user data minus 1 The address plus data length must be less than the end address on the MDS.
data	unsigned char *	Pointer to the buffer in which the data to be written are to be stored
length	uInt	Number of bytes to be written ≥ 0 1 to maximum of 244 bytes
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Windows event due to which the application is waiting for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte of the response telegram with any error codes and interface errors is to be stored (see chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

The MOBY_s_write command may only be sent to the SLG U92 when no command is queued yet on the SLG U92. The antenna must be on. Otherwise an error message is generated.
The command is terminated with an error if the MDS with the specified identification number that is "specifically" called is not located in zone 1.
If no MDS is located in zone 1 and a "non-specific" call occurs, the command waits until an MDS moves into zone 1 or the RESET command arrives.
If there is more than one MDS in zone 1 and a "non-specific" call occurs, the command is terminated with an error.

Notice

With the SLG U92 with ASM 480 (TCP/IP), a maximum of only 228 bytes can be written with one telegram.

3.4.11 Function - moby_s_init

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

You can use this function to "specifically" or "non-specifically" initialize an MDS with a bit pattern. The MDS is located in the antenna field of the SLG U92.

- An initialization call is "specific" when the call contains the identification number of the MDS. Several MDSs may be located in the antenna field. You can determine the identification number of the MDS with the GET function.
- An initialization call is "non-specific" when the call does not contain the identification number of the MDS. Only one MDS may be located in the antenna field.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number or the Ethernet interface
idData	uInt *	Value from 20 to 232 - 1 Identification number of the MDS 0 When an MDS is located in the field and a "non-specific" initialization call is to be performed
setVal	unsigned char	1 byte in hex format with which the entire MDS is to be written 00 hex to FF hex
mdsLength	uInt	Number of bytes on the MDS to be deleted For the number of bytes to be deleted (length of the user data) see the technical description.
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Windows event due to which the application is waiting for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte of the response telegram with any error codes and interface errors is to be stored (see chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

The MOBY_s_init command may only be sent to the SLG U92 when no command is queued yet on the SLG U92. The antenna must be on. Otherwise an error message is generated.
The command is terminated with an error if the MDS with the specified identification number that is "specifically" called is not located in zone 1.
If no MDS is located in zone 1 and a "non-specific" call occurs, the command waits until an MDS moves into zone 1 or the RESET command arrives.
If there is more than one MDS in zone 1 and a "non-specific" call occurs, the command is terminated with an error.

3.4.12 Function - 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);

You can "specifically" copy data with this function. One data area or the contents of the entire data carrier can be copied from one MDS to another. This function permits data to be written (i.e., copied) directly via the user program from one MDS (source) to another MDS (destination).

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number or the Ethernet interface
idData1	uInt	Value from 20 to 232 - 1 Identification number of MDS 1 (source) which is to be copied
addr1	int	0 to maximum length of user data minus 1 Start address of the data to be copied from MDS 1 Address plus data length must be less than the end address.
idData2	uInt	Value from 20 to 232 - 1 Identification number of MDS 2 (destination) to which the data are to be copied
addr2	int	0 to maximum length of user data minus 1 Start address of the data to be copied from MDS 2 Address plus data length must be less than the end address.
len	uInt	Number of the bytes to be copied 1 to maximum length of the user data
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Windows event due to which the application is waiting for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte of the response telegram with any error codes and interface errors is to be stored (see chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

The moby_s_copy command may only be sent to the SLG U92 when no command is yet queued on the SLG U92. The antenna must be on. Otherwise an error message is generated.
If the two specified MDSs are not located in zone 1, the command is terminated with an error.

Notice

The OTP memory cannot be copied with the moby_s_copy command.

3.4.13 Function - 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);

You can use this function to "specifically" or "non-specifically" query status and diagnostic data from an MDS which is located in the antenna field of the SLG U92.

- A read call is "specific" when the call includes the identification number of the MDS. More than one MDS may be located in the antenna field. You can determine the identification number of the MDS with the GET function.
- A read call is "non-specific" when the call does not include the identification number of the MDS. Only one MDS may be located in the antenna field.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number or the Ethernet interface
idData	uInt *	Value from 20 to 232 - 1 Identification number of the MDS 0 When only one MDS is located in the field and a "non-specific" status query is to be made
statusMDS	mobyStatusMDS_t *	Pointer to the structure in which the status and diagnostic information of the response telegram are to be stored
mode	unsigned char	00 hex = Request status and diagnostic data of an MDS
cweek	unsigned char	01 to 35 hex = Current calendar week (1 to 53) for determination of remaining battery life FF hex = No determination of remaining battery life
year	unsigned char	01 to 63 hex = The last two positions of the current year (beginning with 01) for determination of remaining battery life FF hex = No determination of remaining battery life
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Windows event due to which the application is waiting for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte of the response telegram with any error codes and interface errors is to be stored (see chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

The antenna must be on. Otherwise an error message is generated.

Function sequence depends on the two fields "cweek" and "year."

a) If the value in both fields is within the value range, the remaining battery life is output in the response.

b) If one of the values is outside the specified value range, the remaining battery life cannot be calculated and the function is terminated with an error.
If both values are specified as "FF hex" days, the remaining battery life cannot be calculated and the battery life is indicated in the acknowledgment as FFFF hex.

If the MDS that is "specifically" called with its identification number is not located in zone 1, the command is terminated with an error.
If the call is "non-specific" and no MDS is located in zone 1, an error message is generated.
If the call is "non-specific" and more than one MDS is located in zone 1, the command is terminated with an error.

3.4.14 Function - moby_s_readOTP

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

You can use this function to execute a one-time "specific" or "non-specific" read access to the OTP memory (One-Time Programmable) with a length of 16 bytes (= 128 bits).

- A read call is "specific" when the call includes the identification number of the MDS. More than one MDS may be located in the antenna field. You can determine the identification number of the MDS with the GET function.
- A read call is "non-specific" when the call does not include the identification number of the MDS. Only one MDS may be located in the antenna field.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number or the Ethernet interface
idData	uInt *	Value from 20 to 232 - 1 Identification number of the MDS 0 When only one MDS is located in the field and the read call is to be "non-specific"
data	unsigned char *	Pointer to the buffer in which the data read from the OTP memory are to be stored The buffer must have a minimum length of 16 bytes.
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Windows event due to which the application is waiting for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte of the response telegram with any error codes and interface errors is to be stored (see chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

The moby_s_readOTP command may only be sent to the SLG U92 when no command is yet queued on the SLG U92. The antenna must be on. Otherwise an error message is generated.
If the MDS with the identification number specified by the "specific" call is not located in zone 1, the command is terminated with an error.
If there is a "non-specific" call and no MDS is located in zone 1, the command waits until an MDS enters zone 1 or the RESET command arrives.
If there is a "non-specific" call and more than one MDS is located in zone 1, the command is terminated with an error.

Notice

The 128 bits of user information are written to the MDS with the WRITE command. Read accesses must request all 128 bits of information.

3.4.15 Function - moby_s_writeOTP

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

You can use this function to execute a one-time "specific" or "non-specific" write access to the OTP memory (One-Time Programmable) with a length of 16 bytes or 128 bits.

- A write call is "specific" when it includes the identification number of the MDS. More than one MDS may be located in the antenna field. You can determine the identification number of the MDS with the GET function.
- A write call is "non-specific" when it does not include the identification number of the MDS. Only one MDS may be located in the antenna field.

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number or the Ethernet interface
idData	uInt *	Value from 20 to 232 - 1: Identification number of the MDS 0: When only one MDS is located in the field and the write call is to be "non-specific"
data	unsigned char *	Pointer to the buffer in which the data to be written to the OTP memory are stored The buffer must have a minimum length of 16 bytes.
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Windows event due to which the application is waiting for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte of the response telegram with any error codes and interface errors is to be stored (see chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

The moby_s_writeOTP command may only be sent to the SLG U92 when no command is yet queued on the SLG U92. The antenna must be on. Otherwise an error message is generated.
If the MDS with the identification number specified by the "specific" call is not located in zone 1, the command is terminated with an error.
If there is a "non-specific" call and no MDS is located in zone 1, the command waits until an MDS enters zone 1 or the RESET command arrives.
If there is a "non-specific" call and more than one MDS is located in zone 1, the command is terminated with an error.

Notice

The OTP memory can only be written once. This write access must contain all 128 bits of information. A second attempt to write access this memory is rejected with an error message.

3.5 DI/DO Functions

The DI/DO functions moby_readDE and moby_writeDA can be used to read (scan) digital inputs (DI) and write (set) digital outputs (DO). They can only be used with the SLG on the ASM 420 and SIM 41.

3.5.1 Function - moby_readDE

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

This function reads digital inputs (DI).

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number
deda	mobyDEDA_t *	Pointer to the structure in which the read signal states are stored
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Windows event due to which the application is waiting for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte of the response telegram with any error codes and interface errors is to be stored (see chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

After the call, the signal states of the digital inputs are available in the buffer. The number and type of representation vary depending on the particular MOBY system. See documentation /01/ or /02/ of your MOBY.

3.5.2 Function - moby_writeDA

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

This function sets (writes) the digital outputs (DO).

Parameter	Type	Description
handle	mobyHandle_t	Handle of the serial interface and the channel number
deda	mobyDEDA_t *	Pointer to the buffer in which the signal states to be written are stored
sync	HANDLE	Handle for synchronization with the response telegram. Initialized Windows event due to which the application is waiting for the response telegram.
err	mobyErr_t *	Pointer to the structure in which the status byte of the response telegram with any error codes and interface errors is to be stored (see chapter 3.7.2).

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Notice

The buffer contains the signal states of the digital outputs to be set and the related conditions. The number and type of representation vary depending on the particular MOBY system. See documentation /01/ or /02/ of your MOBY.

3.6 Function - moby_version

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

The moby_version function is used to scan the version of the dynamic link library MOBY_API.DLL or MOBY_API_T.DLL. The version number is returned in two parts - the main number xx and the subnumber yy. It is combined as "xx.yy."

Parameter	Type	Description
major	int *	Memory word for the value xx of the version number
minor	int *	Memory word for the value yy of the version number

Return value:

≥ 0	No error, command executed.
< 0	Interface error. See chapter 3.7.1.

Example: Version 1.0 appears as xx = 1 and yy = 0.

3.7 Function Errors

3.7.1 Interface Error as Return Value

The interface errors supplied as return values by the described functions are listed below.

- Errors directly from the MOBY_API or MOBY_API_T DLL
- Errors from the 3964R driver (only serial link to PC)
- Errors from the SLG, ASM with SLG or SLA or SIM

The errors from the 3964R driver are divided into two classes.

• Status error messages:	These messages signal status errors which occur while telegrams are being transferred. They are usually related to the interface or local system resources.
• System error messages:	These messages indicate that call functions malfunctioned. They usually mean a fatal error in the system and/or the configuration settings.

The return value is type "long integer" and is always negative. Its layout is shown below.

Return value = Error code **OR** error block **OR** error number

The values of error code, error block and error number are OR-linked and may assume the following values.

Error code:	0x80 00 00 00	
Error block:	0x1 00 00	Error directly from the MOBY_API DLL
	0x2 00 00	Error from the 3964R driver
	0x3 00 00	Error from the SLG, ASM with SLG or SLA or SIM

Error numbers:

- MOBY_API or MOBY_API_T

0x00 00	Fatal error
0x00 01	Wrong device handle
0x00 02	Wrong parameter
0x00 03	Wrong/unsuitable response telegram
0x00 04	Response telegram has wrong length.
0x00 05	Match buffer full
0x00 06	Wrong MDS type
0x00 07	No other handle available
0x00 08	Windows system error
0x00 09	Command terminated with "moby_stop"
0x00 0A	No ID numbers with MOBY I
0x00 0B	Handle still open
0x00 0C	Command terminated by error (implicit "moby_stop")
0x00 0D	Channel already open
0x00 0E	Channel already started
0x00 0F	Channel not yet started
0x00 10	Last status query not yet concluded
0x00 11	This number (MOBY_ERR_NOTSUPPORTED) stands for functions not covered by the selected interface (e. g., moby_s_getID, when there is no MOBY U SLG)
0x00 12	Wrong IP address assigned
0x00 13	Error in socket handle
0x00 14	Error in TCP/IP connection establishment
0x00 15	Invalid device ID

When one of the error numbers 0x00 00, 0x00 01, 0x00 02, 0x00 05, 0x00 07, 0x00 12, 0x00 13, 0x00 14 or 0x00 15 is reported, the applicable interface must be closed with moby_close (if the interface is open) and then reopened with moby_open.

- 3964R driver

Status error messages

0x80 01	Too many jobs, sending queue full
0x80 02	Attempt to establish connection failed (send)
0x80 04	Telegram transfer failed (send)
0x80 08	Receiving buffer too small for receipt
0x80 10	Block check error while receiving
0x80 20	Timeout during connection establishment (send)
0x80 40	Timeout during a telegram (receive)
0x80 80	Timeout during connection disconnection (send)
0x81 00	COM message: break
0x82 00	COM message: parity error
0x84 00	COM message: framing error
0x88 00	COM message: Overrun
0x90 00	Duplex conflict (send)
0xC0 00	Unknown system error

System error messages

0xA0 00	No error
0xA0 02	Transferred handle invalid
0xA0 03	Not enough memory available
0xA0 04	No room in the sending buffer
0xA0 05	Buffer overflow
0xA0 06	Error during operation on a global event (OS error)
0xA0 07	Interface already open
0xA0 08	Interface not open
0xA0 09	Not used at this time
0xA0 0A	Not used at this time
0xA0 0B	Not used at this time
0xA0 0C	Not used at this time
0xA0 0D	Not used at this time
0xA0 0E	Not used at this time
0xA0 0F	Not used at this time
0xA0 10	Not used at this time
0xA0 11	Message parameter invalid
0xA0 12	Not used at this time
0xA0 13	Faulty receive job
0xA0 14	Faulty send job
0xA0 15	Not used at this time
0xA0 16	Not used at this time
0xA0 17	NAK while sending a telegram
0xA0 18	Character received while sending a telegram (not NAK)
0xA0 64	Interface initialization error
0xA0 65	Problems with thread or with opening/closing the COM interface
0xA0 66	Not used at this time
0xA0 67	Not used at this time
0xA0 68	Not used at this time
0xA0 69	Not used at this time
0xE0 01	Addressed port not configured
0xE0 02	Error while accessing the Windows registry
0xE0 03	No more ports to open

• SLG, ASM or SIM	For possible error numbers, see chapter 3.7.2 under MOBY error codes.

Notice

When Windows interrupts communication between PC and the ASM, SLG or SIM because a monitoring time is exceeded (see chapter 2.2.3), error 0x000C can occur on the MOBY API C interface or error 0xA017 can occur directly on the 3964R driver. An unexpected telegram may also occur which is not reported unless the callback routine is used. See chapters 3.1.3 and 3.3.12. This unexpected telegram must be rejected.

- After error 0x000C has occurred the moby_start function must be called and the failed function must then be repeated. The moby_stop function is not necessary. If you call this function anyway, error 0x000F occurs.
- Following error 0xA017 the command during which the error occurred can be repeated directly without moby_stop and moby_start.

If an error occurs during moby_start, close the COM interface with moby_close and then open it again.

3.7.2 MOBY Status

After a MOBY function is executed (response telegram has arrived), the MOBY status byte of the response telegram is stored under the "mobyErr_t" union addressed with the "status" pointer. The status byte can be addressed as

- Structure "mobyStatus_t" or
- Long integer variable "error"

(See chapter 4.1 for header file MOBY_API.H or chapter 4.3 for header file MOBY_API_T.H.)
The error and status information can be read from the individual structure parameters (by bit) or as a unit (see also error numbers from the ASM or SIM with SLG or SLA in chapter 3.7.1).

"mobyStatus_t" structure

Table 3-2 Structure parameters of MOBY status

Structure Parameter (Variable)	Meaning
error	Error identifier
dummy	In reserve
busyASM	MDS command on ASM active/no MDS on ASM active
anwMDS	Presence message
batMDS	Status of MDS battery voltage (battery 1)
diagBatMDS507	Status of battery voltage, MDS 507/407 E (battery 2)
eccDone	ECC offset status on MDS
errorCode	MOBY error code

error

Error identifier

1:	Error: "errorCode" > 0
0:	No error

dummy

Area in reserve (bits)

busyASM

MDS command on ASM active/no MDS command on ASM active (corresponds to bit 0 in ANW/Busy byte)

1:	MDS command on ASM active
0:	No MDS command on ASM active

anwMDS

The presence message corresponds to bit 1 in the ANW/Busy byte. It indicates whether an MDS is located in the field of the SLG/SLA.

0:	No MDS in the field
1:	MDS in the field

batMDS

The battery voltage status corresponds to bit 7 of the status byte (battery 1). It indicates the status of the dialog battery on the MDS.

1:	Battery on MDS has dropped below threshold value. This bit is always set for MDS types with EEPROM memory.

diagBatMDS507

The dialog battery status on the MDS corresponds to bit 6 of the status byte (battery 2). It only applies to MDS 507/407 E.

1:	Battery under threshold value With other MDSs, the bit can be "0" or "1."

eccDone

The ECC offset status corresponds to bit 5 of the status byte. It only applies when "with ECC offset" was specified for the moby_start system function.

1:	ECC offset was performed. (The data in the result telegram are okay.)

errorCode

The MOBY error code corresponds to bits 0 to 4 in the MOBY status byte.

Table 3-3 MOBY error code

Error (Hex)	Meaning	Assignment to ASM/SIM/SLG					
		ASM 824	ASM 724	ASM 424	ASM 420	SIM 41	SLG U92
00	No error. Response telegram correct.	x	x	x	x	x	x
01	ANW error: MDS left field while command active	x	x	x	x	x	x
02	ANW error: MDS passed SLG without command	x	x	x	x	x	–
	A queued MDS command was terminated by the “turn off antenna” command.	–	–	–	–	–	x
03	Error in connection to the SLG/SIM/SLA	x	x	x	x	x	–
04	Error in memory of MDS (not initialized)	x	x	x	x	x	x
05	Command from SLG/ASM cannot be interpreted.	x	x	x	x	x	x
06	Field interference on SLG/SIM/SLA	x	x	x	x	x	x
07	Too many sending errors	x	x	x	x	x	–
08	MDS reports CRC errors very frequently	x	x	x	–	x	–
09	INIT: CRC error	x	x	x	x	x	–
0A	INIT: MDS cannot be initialized.	x	x	x	x	x	–

Table 3-3 MOBY error code

Error (Hex)	Meaning	Assignment to ASM/SIM/SLG					
		ASM 824	ASM 724	ASM 424	ASM 420	SIM 41	SLG U92
0B	INIT: Timeout during initialization	x	x	x	x	x	–
	MDS memory cannot be read correctly	–	–	–	–	–	x
0C	INIT: Write error during initialization	x	x	x	x	x	x
	Repeated storing on OTP memory	–	–	–	–	–	x
0D	Address error	x	x	x	x	x	x
0E	ECC mode: Data on MDS are wrong.	x	x	x	x	x	–
0F	RESET message after return of power	x	x	x	x	x	–
10	NEXT command is illegal.	x	x	x	x	x	x
11	Short circuit or overload on the digital outputs	x	x	x	–	–	–
12	Internal firmware error	x	x	x	–	–	–
13	Watchdog	x	x	x	–	–	–
	There is not enough buffer space on the SLG to store the command.	–	–	–	–	–	x
14	Firmware error	x	x	x	–	–	x
	Watchdog message from SLG U	–	–	–	–	–	x
15	Parameter assignment error	x	x	x	–	–	x
16	Unsuitable connection configuration	x	x	x	–	–	–
17	Protocol error	x	x	x	–	–	–
18	Only RESET command permitted	x	x	x	–	–	x
19	Previous command is active.	–	–	–	x	x	x
	Buffer overflow, telegram buffer full	x	x	x	–	–	–
1A	3964R error (connection interrupted)	x	x	x	–	–	–
1B	Sending job on SLG repeated too often. Data loss possible. RESET command required.	–	–	–	–	–	x
1C	Turn antenna on/off	–	–	–	–	–	x

Table 3-3 MOBY error code

Error (Hex)	Meaning	Assignment to ASM/SIM/SLG					
		ASM 824	ASM 724	ASM 424	ASM 420	SIM 41	SLG U92
1D	Not enough RAM on the MDS	–	–	–	x	–	–
	Number of MDSs > bunch	–	–	–	–	–	x
1E	Wrong number of characters in the telegram	x	x	x	x	x	x
1F	SLG/ASM command deleted with RESET	x	x	x	x	x	x

Long integer variable "error"

When the error and status information is read as a long integer value, the individual pieces of information must be evaluated by masking.

4 Header Files

The header file

- MOBY_API.H for serial link to PC or
- MOBY_API_T.H for link to Ethernet

must be integrated with the "#include" preprocessor command. This declares all function calls and constants.

4.1 Header File – 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 Header File – 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 Header File – 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 Sample Application

To make it even easier to get started with the implementation of a user application, we have also included a sample application in source code in addition to the C-library for each type of link (serial and Ethernet). The sample applications can also be used as an executable program.

5.1 Sample Application in Source Code for Serial Link to PC

The sample application is an executable program for the SLA 81 on the ASM 824 as READ system with MDS F4xx. The ASM 824 must be run on the COM2 interface, and the SLA 81 must use channel 4 of the ASM 824.
In addition, the source code of this sample application can be used as the READ system for SLA 81 on ASM 824 with MDS F1xx or SLA 71 on ASM 724 by adding the appropriate define instruction.

```
// 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 telegrams
// 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 Sample Application in Source Code for Link to Ethernet

A sample application in source code - TEST.CPP (VC++ 6.0) - is available for a simple READ application for the SLG U92 on the ASM 480. Before the application is compiled, the IP address "157.163.170.2" predefined in TEST.CPP must be adjusted to the actual IP address.

```
// 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 Description of Communication to the ASM 424/724/824 with 3964R Protocol

This appendix is written for users who want to place their application directly on the operating system or driver level and do not want to use the MOBY API C-library.

For more information on the 3964R driver level, see also the user's manual on using the 3964R protocol under Windows NT 4.0/95 (included as PDF file on the MOBY software CD).

A.1 General

The ASM 424/724/824 interface modules use a Baud rate of 9.6 kBaud, 19.2 kBaud or 38.4 kBaud. The Baud rate is recognized automatically.
The data format consists of 1 start bit, 8 data bits, 1 parity bit (parity supplements to give an odd total number of ones), and 1 stop bit.
After the module is turned on (startup), the ASM attempts to send a startup telegram during which the 3 different Baud rates are tested up to two times. When the telegram is sent successfully, that Baud rate is considered "recognized." If not, the ASM waits for a telegram (STX) and then acknowledges with DLE during the second STX when the Baud rate is found. The Baud rate cannot be changed during operation.

Notice

When a telegram cannot be sent due to a transmission error, all four channels are reset (i.e., the commands are deleted).

A.2 Protocol Settings

The ASM is permanently parameterized to master when an initialization conflict occurs.
The 3964R driver of the ASM uses the following settings.

• Attempts to establish the connection:	3
• Attempts to make block transmission:	6
• Acknowledgment delay time:	2 sec
• Character delay time:	220 msec
• Block wait time:	10 sec
• Wait time for subsequent telegrams to be sent:	10 to 20 msec

The type of interface can be set with the parameterization switch (8).

Up:	RS 422
Down:	RS 232

The other parameters are transferred with the RESET command.

A.3 LEDs on the 3964R Interface Side of the ASM

Table A-1 LEDs on the ASM

LED	Remarks
ON (green)	Power-on LED (ASM is powered.)
ACT (green)	This LED flashes once briefly to indicate that a command has been executed successfully (not for status commands).
SF/BF (red)	• Continuously on during startup • Off when telegram is correctly received or sent • Flashes (once to twice per second) when a send or receive error occurs (telegram could not be transmitted in acc. w. 3964R protocol).

A.4 General Communication Procedure

The ASM will not accept a command unless the previous command has been concluded. The only exceptions are the RESET and status commands. These two commands may occur simultaneously. RESET and status commands are processed immediately.

Notice

When two status commands are sent in succession but there is no acknowledgment response message from the first command, only one acknowledgment for the last status command is output. With the RESET command, you should always wait for the acknowledgment before sending a new command.

A.5 Overview of Commands

Table A-2 List of the commands

Command	Code	Description
RESET	00	Resets both the active command and all other unprocessed commands in the buffer of the affected channel (command termination). Default setting of the ASM mode. In detail • Operation with or without ANW or presence control • MOBY operating mode (E/F or I)
READ WRITE INIT	50 51 18	Read data from MDS (40: with ECC) Write data to MDS (41: with ECC) Initialize data memory (1A: with ECC)
STATUS	01	Status query of ASM • Presence. Command active. • Status of ASM

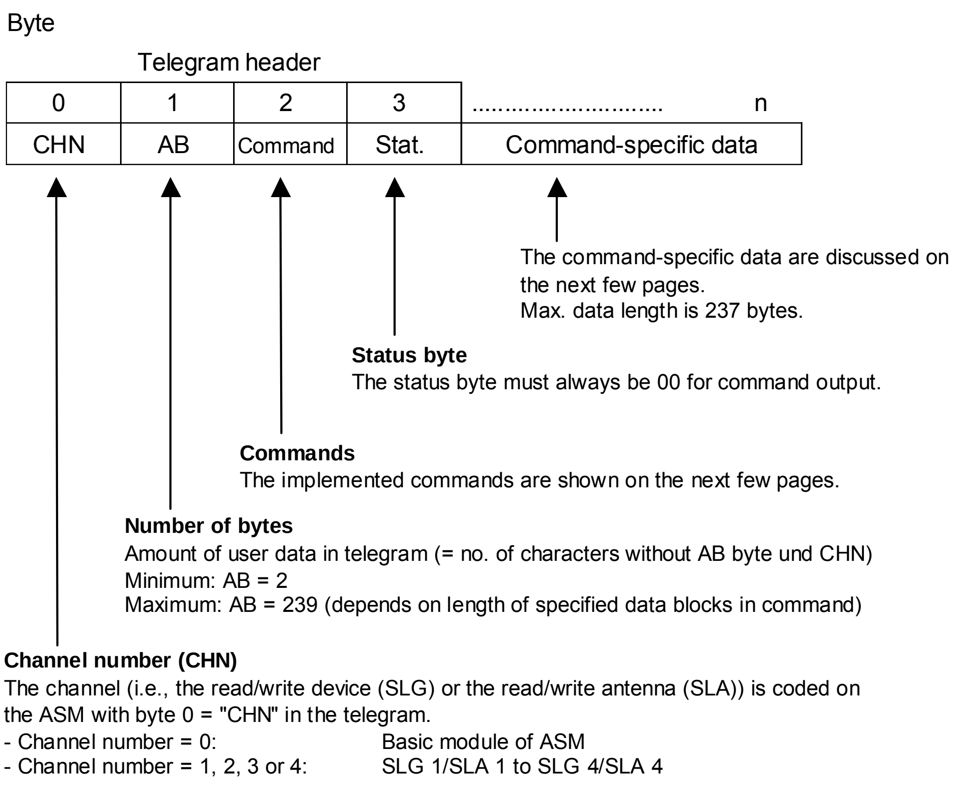
Table A-2 List of the commands

Command	Code	Description
NEXT	07	NEXT command The SLG has finished processing the command for the MDS in its field. The next MDS entering the field is processed with command started after NEXT. Prerequisite: presence control is activated.

Notice

ECC mode can only be used with MOBY I with the ASM 424.

General telegram layout



The command-specific data are discussed on the next few pages.
Max. data length is 237 bytes.

Status byte
The status byte must always be 00 for command output.

Commands
The implemented commands are shown on the next few pages.

Number of bytes
Amount of user data in telegram (= no. of characters without AB byte und CHN)
Minimum: AB = 2
Maximum: AB = 239 (depends on length of specified data blocks in command)

Channel number (CHN)
The channel (i.e., the read/write device (SLG) or the read/write antenna (SLA)) is coded on the ASM with byte 0 = "CHN" in the telegram.

- Channel number = 0:	Basic module of ASM
- Channel number = 1, 2, 3 or 4:	SLG 1/SLA 1 to SLG 4/SLA 4

A.6 Telegram Layout of the Commands/Acknowledgments to/from the ASM

The maximum length of the telegram is 241 bytes, including the 7-byte telegram header.

Data format

When not otherwise noted, all numbers are specified in hex format.

Channel number - CHN

Byte 0 = CHN of the telegram contains the channel on the ASM. The channel is actually the read/write device (SLG) or the read/write antenna (SLA).

• Channel number = 0:	Basic module of the ASM
• Channel number = 1, 2, 3 or 4:	SLG 1/SLA 1 to SLG 4/SLA 4

A.6.1 Startup Telegram

Byte

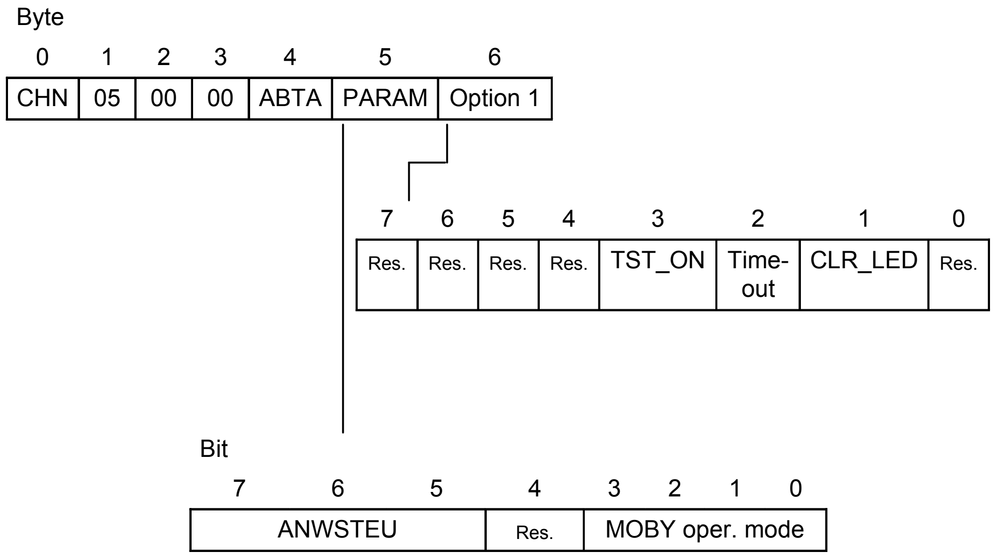
0	1	2	3
CHN	02	00	0F

CHN 00 Hex = Basic module of the ASM

The startup telegram arrives as an automatic "acknowledgment" after the hardware starts up. The CHN is 0 since it is only sent once by the "total device" and not by each channel.

A.6.2 RESET

Command



CHN	01, 02, 03 or 04 Hex = SLG 1/SLA 1 to SLG 4/SLA 4	
ABTA	Setting of the scanning time for MOBY I-Long-Range with ASM 424	
PARAM	**ANWSTEU**	
	000 =	Operation without ANW control, without ANW monitoring
	001 =	Operation without ANW control, with ANW monitoring [1)]
	010 =	Operation with ANW control and ANW monitoring via the firmware [2)]

1) Operating modes with MDS F1xx (MOBY F, read only) must always be parameterized with ANW monitoring.

2) ANW control only applies to ASM 424. Do not use for ASM 824 with MDS F1xx and ASM 724.

MOBY operating mode

The following MOBY operating modes can be set on your ASM.

MOBY I, ASM 424:

1 =	MOBY I/MOBY E (SLG 7x)
2 =	Reserved
4 =	MOBY I with MDS 507
8 =	Reserved for MOBY I dialog
9 =	Reserved for MOBY V
A =	MOBY F: with MDS F1xx
B =	MOBY F: with MDS F4xx
C =	MOBY F: with MDS F2xx

MOBY E, ASM 724:

0 =	All parameters in the RESET telegram are ignored (default setting).
1 =	MOBY E (SLA 71)

MOBY F, ASM 824:

0 =	All parameters in the RESET telegram are ignored (default setting).
A =	MOBY F (SLA 81) and MDS F1xx (read only)
B =	MOBY F (SLA 81) and MDS F4xx (read/write)

Option 1

TST_ON = 1:

ASM responds with error if field interference on SLG (only for ASM 424).

Timeout = 1:

ASM responds with error if no MDS is in its field (only for ASM 424).

CLR_LED = 1:

ERROR_LED (channel-specific) is cleared.

Notice

The parameters in the RESET telegram are only used for the first telegram after power ON (i.e., parameter changes do not take effect until the power is turned off and on again).
All channels must be parameterized with the same MDS. Mixing is not possible.

Acknowledgment

Byte

0	1	2	3	4	5	6
CHN	05	00	Status	High	Low	In res. 1

or

0	1	2	3
CHN	02	00	Status

CHN	01, 02, 03 or 04 Hex = SLG 1/SLA 1 to SLG 4/SLA 4
Status	MOBY status
High	Firmware status in ASCII
Low	Firmware status in ASCII
Res. 1	Reserved

A.6.3 WRITE

Command

Byte

0	1	2	3	4	5	6	Starting with 7
CHN	AB	51*)	00	ADR H	ADR L	Length	Data

*) With ECC: Command = 41

CHN	01, 02, 03 or 04 Hex = SLG 1/SLA 1 to SLG 4/SLA 4
AB	Number of subsequent characters in the telegram
ADR H	Start address on MDS (more significant portion of address)
ADR L	Start address on MDS (less significant portion of address)
Length	Length of the data block (max. of 234 bytes)
Data	Data to be written

Acknowledgment

Byte

0	1	2	3
CHN	02	51*)	Status

*) With ECC: Command = 41

CHN	01, 02, 03 or 04 Hex = SLG 1/SLA 1 to SLG 4/SLA 4
Status	MOBY status

A.6.4 READ

Command

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 or 04 Hex = SLG 1/SLA 1 to SLG 4/SLA 4
ADR H	Start address on MDS (more significant portion of address)
ADR L	Start address on MDS (less significant portion of address)
Length	Length of the data to be read (max. of 234 bytes)

Acknowledgment

Byte

0	1	2	3	4	5	6	7 to ...
CHN	AB	50*)	Status	ADR H	ADR L	Length	Data

*) With ECC: Command = 40

CHN	01, 02, 03 or 04 Hex = SLG 1/SLA 1 to SLG 4/SLA 4
AB	Number of subsequent characters in the telegram
Status	MOBY status
A ADR H	Start address on MDS (more significant portion of address)
ADR L	Start address on MDS (less significant portion of address)
Length	Length of the data block (data which were read)
Data	Data which were read

A.6.5 INIT

Command

Byte

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

*) With ECC: Command = 1A

CHN	01, 02, 03 or 04 Hex = SLG 1/SLA 1 to SLG 4/SLA 4
INIT pattern	During initialization, the MDS is written with the value "INIT pattern."
ADR H	Amount of memory to be allocated (more significant portion of the address)
ADR L	Amount of memory to be allocated (less significant portion of the address)

Acknowledgment

Byte

0	1	2	3
CHN	02	18*)	Status

*) With ECC: Command = 1A

CHN	01, 02, 03 or 04 Hex = SLG 1/SLA 1 to SLG 4/SLA 4
Status	MOBY status

A.6.6 STATUS

Command

Byte

0	1	2	3
CHN	02	01	00

CHN 01, 02, 03 or 04 Hex = SLG 1/SLA 1 to SLG 4/SLA 4

Acknowledgment

Byte

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

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

CHN	01, 02, 03 or 04 Hex = SLG 1/SLA 1 to SLG 4/SLA 4
Status	ERR_LED indicator
ANW	Data memory in field
Busy	Command being processed
Res.	00 Hex (n reserve)

A.6.7 NEXT

Command

Byte

0	1	2	3
CHN	02	07	00

CHN	01, 02, 03 or 04 Hex = SLG 1/SLA 1 to SLG 4/SLA 4

Acknowledgment

Byte

0	1	2	3
CHN	02	07	Status

CHN	01, 02, 03 or 04 Hex = SLG 1/SLA 1 to SLG 4/SLA 4
Status	MOBY status

A.7 MOBY F – Special Features in Read-Only Mode (Only ASM 824/SLA 81 with MDS F1xx)

The RESET, STATUS and READ commands are only permitted in a certain format for read-only use.
As with READ/WRITE operation, only one command is possible at a time but RESET and STATUS can always be issued. After a startup and RESET command, all data memories are read and stored in the telegram buffer (maximum of 50 telegrams) if not already fetched with a READ command.
When the buffer is full, all other data memories are ignored. The last telegram contains an error message.

When a new data memory (with different data) is read, the next telegram can be fetched from the ASM.

Command

READ Command to trigger send of ASM 824

Byte

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

CHN 01, 02, 03 or 04 Hex = SLG 1/SLA 1 to SLG 4/SLA 4

Acknowledgment

Byte

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

HR/ANW

Bit

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

CHN	01, 02, 03 or 04 Hex = SLG 1/SLA 1 to SLG 4/SLA 4
Status	MOBY status
HR	Historical read. There was just enough time to generate the data (e.g., due to high-speed data memory).
ANW	0 = MDS not present 1 = MDS present
Counter	Is incremented by 1 for each new telegram (00 to FF Hex)
ID number	Identification number of MDS F1xx (5 bytes)

The next command is not a normal READ command as described in A.6.4. This command is used to control data flow. The data were acquired automatically with the ASM and must be read in with this command.
This command can be used to fetch an existing read-only telegram. If no telegram exists, the ASM sends one as soon as a new one arrives.

A.8 Mobile Data Memories

Types of memories

Mobile data memories with different memories are available to the user.

Table A-3 Types and sizes of memory

System	Memory Size Normal	Memory Size with ECC	Memory Type	MDS Type
MOBY I	62 bytes	42 bytes*	RAM	Ex.: MDS 114
MOBY I	128 bytes	112 bytes*	EEPROM	Ex.: MDS 213 E
MOBY I	2 Kbytes	1,7 bytes*	RAM	Ex.: MDS 302
MOBY I	8 Kbytes	7 Kbytes*	FRAM	Ex.: MDS 401
MOBY I	32 Kbytes	28 Kbytes*	RAM	Ex.: MDS 506
MOBY E	752 bytes	–[1]	EEPROM	MDS E6xx
MOBY F	5 bytes	–[1]	Fixed code	MDS F1xx
MOBY F	32 bytes	–[1]	EEPROM	MDS F2xx
MOBY F	256 bytes	–[1]	EEPROM	MDS F4xx

* *Net capacity in ECC mode*

1 *ECC mode cannot be used with MOBY E/F.*

The following table lists the address areas of the individual MDS models.

Table A-4 Address area of MDS

Addressing	Hexadecimal, 16 Bits		Fixed Point Number, 16 Bits	
	Normal	With ECC	Normal	With ECC
MOBY I	**62-byte data memory with RAM**			
Start address	0000	0000	+0	+0
End address	003D	0029	+61	+41
MOBY I	**128-byte data memory with EEPROM**			
Start address	0000	0000	+0	+0
End address	007F	006F	+127	+111
MOBY I	**2-Kbyte data memory with RAM**			
Start address	0000	0000	+0	+0
End address	07FC	06F1	+2044	+1777
MOBY I	**8-Kbyte data memory with EEPROM**			
Start address	0000	0000	+0	+0
End address	1FFC	1BF1	+8188	+7153
MOBY I	**32-Kbyte data memory with RAM**			
Start address	0000	0000	+0	+0
End address	7FFC	6FF1	+32764	+28657
MOBY E	**752-byte data memory with EEPROM**			
Start address	0000	–[1]	+0	–[1]
End address	02EF	–[1]	751	–[1]
Read serial number for MOBY E *				
Start address	1FF0	–[1]	8176	–[1]
Length	4	–[1]	4	–[1]
MOBY E	**5-byte MDS F1xx (fixed code)**			
Start address	0000	–[1]	+0	–[1]
End address	0004	–[1]	+4	–[1]
MOBY F	**32-byte MDS F2xx EEPROM**			
Start address	0010	–[1]	+16	–[1]
End address	001F	–[1]	+31	–[1]
ID no. (can only be read as a whole)				
Start address	0000	–[1]	+0	–[1]
Length	4	–[1]	+4	–[1]
MOBY F	**192-byte MDS F4xx EEPROM**			
Start address	0040	–[1]	+64	–[1]
End address	00FF	–[1]	+255	–[1]
ID no. (can only be read as a whole)				
Start address	0000	–[1]	+0	–[1]
Length	4	–[1]	+4	–[1]

* *Data representation in DATDB: 1st byte = MSB, 4th byte = LSB*

1 *ECC mode cannot be used with MOBY E/F.*

The data memories are addressed as shown in the table.

A.9 Status and Error Codes (ASM 424, ASM 724 and ASM 824)

Table A-5 Status and error codes of ASM 424/724/824

Error Code (Hex)	Flashing LED (per Channel)	Description
00	00	No error
–	01	Same as 0F
01	02	Presence error. Command only partially executed. With ASM 824 (MDS F1xx): 2 data memories in field at once
02	02	Presence error. Data memory wasn't processed. Timeout for command execution (MDS on field boundary).
03	03	Error in connection to SLG/SLA
04	04	MDS RAM error (not initialized)
05	05	Unknown command
06	06	Field interference on SLG (MOBY F SLA 81: INIT error)
07	07	Too many sending errors
08	08	CRC send error
09	09	CRC error during acknowledgment receipt (only during initialization)
0A	10	MDS refuses initialization.
0B	11	Timeout during initialization
0C	12	MDS memory cannot be written.
0D	13	Address error on MDS MDS on boundary of field
0E	14	ECC error
0F	01	Startup message
10	16	NEXT command not possible or not permitted
12	18	Internal firmware error
13	19	Watchdog
14	20	Firmware error (checksum error telegram, stack overflow, change in program code, timeout for channel connections, and so on)
15	21	Parameterization error
16	22	Connection configuration unsuitable
17	23	Protocol error
18	–	Only RESET command permitted
19	25	Buffer overflow, entire telegram buffer full. Previous command(s) active.
1A	–	PROFIBUS or 3964 error (bus link was interrupted)
1E	30	Telegram structure is wrong.
1F	–	Command or commands were terminated with RESET.
20 (binary xx1x xxxx)	32	**Not an error message!** Only occurs when enabled ECC driver is being used. It indicates that the driver has detected a 1-bit error and corrected it. The read or write data are okay.

Table A-5 Status and error codes of ASM 424/724/824

Error Code (Hex)	**Flashing LED (per Channel)**	**Description**
40 (binary x1xx xxxx)	64	**Not an error message!** This bit is usually always set. It is reserved for indication of the status of a 2nd battery on the MDS
80 (binary 1xxx xxxx)	128	**Not an error message!** Battery voltage of the MDS has dropped below the threshold value. We recommend replacing the MDS immediately. With MDS types with EEPROM, this status bit is always set. With SINUMERIK, the battery message does not have the ID “F” in IDENTIFICATION. To detect a weak battery, the “fnr” field can be evaluated at some point in the total system.

B Programming the SLG U92 Based on the Operating System or 3964R Driver

For whom is this appendix written?

This appendix is written for users who place their applications directly on the operating system or the 3964R driver level and do not use the MOBY API C library. Additional information on the 3964R driver level is available from the user's guide "3964R protocol under Windows NT 4.0/95" (see PDF file on the "Software MOBY" CD) /07/.

Notice

For applications which are not based on the MOBY API C interface and are not directly based on the 3964R driver of the MOBY API C library, an appropriate 3964R driver for the target hardware (serial link) must be used which meets the requirements of the SLG U92. For the behavior of the 3964R driver on the SLG U92, see MOBY U documentation /06/.

B.1 General Information on Communication of the SLG U92

MDS and system functions are available for communication with the mobile data memories (MDS U313, MDS U315, MDS U524, MDS U525 and MDS U589) and for control of the system behavior of the SLG U92. Management of the data on the mobile data memories (MDS U313, MDS U315, MDS U524, MDS U525 and MDS U589) uses byte addressing with absolute addresses (“normal addressing”).

You can choose between three versions of the system control.

1. MOBY I call-compatible
   - Read/write distance up to 1.5 m ⇒ Range limited to 1.5 m (fixed setting)
   - Bunch/multitag = 1 ⇒ Bunch set to 1 (fixed)
   - Without proximity switch operation
     The SLG U92 does not evaluate the digital inputs on the service interface.
   - Without SLG synchronization
2. MOBY I call-compatible with expanded commands
   - Read/write distance of 0.5 m up to maximum of 3.5 m, can be set in increments of 0.5 m.
   - Bunch/multitag = 1 ⇒ Bunch set to 1 (fixed)
   - Proximity switch possible
   - SLG synchronization possible

3. MOBY U whith multitag processing
   - Read/write range adjustable in 0.5-m increments from 0.5 m to maximum 3.5 m
   - Bunch/multitag ≤ 12
   - Proximity switch possible
   - SLG synchronization possible

You can select which version by making the appropriate setting in the RESET system telegram (see appendices B.2.1.2.1, B.3.1.2.1 and B.4.1.2.1).

The functions are sent to the SLG U92 as telegrams with the 3964R protocol.
An acknowledgment with or without user data is returned by the SLG U92 for each telegram received. In addition, messages may arrive non-cyclically from the SLG U92.
The telegrams are always comprised of a telegram header and, depending on the function, no user data or up to 251 bytes of user data. A telegram can have up to 254 bytes.

Telegram structure

	Telegram Header			**User Data (Max. of 251 Bytes)**
Byte	0	1	2	3 to max. of 253 bytes
	AB [hex]	Com-mand [hex]	Status [hex]	User data [hex]

AB	=	Telegram length in bytes without the AB byte
Command	=	Function ID
Status	=	Status field "Status"
User data	=	Parameters to write data to MDS, …, read data from MDS, diagnostic data, status data, …

The 3964R driver must be configured as shown below for communication with the SLG U92.

• Data bits	8
• Stop bits	1
• Parity	Odd
• Send buffer	255
• Receive buffer	255
• Baud rate	19200, 38400, 57600 or 115200 baud
• SLG U92	Slave with automatic baud rate recognition

B.2 MOBY I Call-Compatible (Version 1)

B.2.1 Telegrams to the SLG U92

MDS functions

• INIT	Initialize MDS
• WRITE	Write data block
• READ	Read data block

System functions

• RESET	Reset SLG
• SLG-STATUS	SLG status/diagnosis
• L-UEB	Cable monitoring

The RESET command resets the SLG U92 to a defined state. You determine how the SLG U92 system reacts by specifying the parameters of this command.

Telegram overview

	Telegram Header			User Data (Max. of 251 Bytes)			
Byte	0	1	2	3 to max. of 253 bytes			
Telegrams (Function)	AB [hex]	Com-mand [hex]	Status [hex]	User data [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	=	Telegram length in bytes without the AB byte
ABL	=	Variable telegram length in bytes without the AB byte, depending on the length parameter ⇒ 5 + length

Notice

The data are shown in the telegram overview and in the following individual telegram presentations in hexadecimal format (hex).

B.2.1.1 MDS Functions

The MDS functions INIT, READ and WRITE read or write data from/to the MDS.

B.2.1.1.1 INIT Function

The INIT function is used to initialize with a bit pattern the MDS which is located in the antenna field of the SLG U92. It is an "untargeted" initialization call since the MDS is not identified with the ID number.

Byte	0	1	2	3	4	5 6
Parameter	06	03	00	Date	00	Length

Date	Binary value	Bit pattern 00 hex up to FF hex, with which the data carrier is to be initialized (written).		
Length	Binary value	32768	=	Length in bytes of the data memory MDS U524, MDS U525 and MDS U589
		2048	=	Length in bytes of the MDS U313 and MDS U315 data memory

The INIT command may only be used for the SLG U92 and only when no command is queued on the SLG U92. The antenna must be on. Otherwise an error message is generated.
When more than one MDS is located in zone 1, the command is terminated with an error.
When no MDS is located in zone 1, the command waits until an MDS enters zone 1 or the RESET command arrives.

B.2.1.1.2 WRITE Function

The WRITE function is used to write data to the MDS which is located in the antenna field of the SLG U92. It is an "untargeted" write call since the MDS is not identified by the ID number.

Byte	0	1	2	3 4	5	6 Up to 253
Parameter	ABL	01	00	Address	Length	Data

ABL	Binary value	1 to 253	=	Telegram length in bytes without the AB byte
Address	Binary value	0 to maximum length of the user data minus 1. Start address on the MDS for the data to be written. The address plus data length must be less than the end address.		
		- 16	=	FFF0 hex Start address of the OTP memory
Length	Binary value	1 to 248	=	Length of the user data to be written. Start address + length must be less than value of the memory length of the MDS in bytes minus 1.
		16	=	Length of the OTP memory
Data	Binary info	User data to be written to the MDS		

The WRITE command may only be used for the SLG U92 and then only when no command is queued on the SLG U92. The antenna must be on. Otherwise an error message is generated.
When more than one MDS is located in zone 1, the command is terminated with an error.
When no MDS is located in zone 1, the command waits until an MDS enters zone 1 or a RESET command arrives.

Notice

The 128 bits of user information in the OTP memory are addressed with the start address –16 (FFF0 hex). The OTP memory can only be written once. The write command must transfer all 128 bits of information at one time. A second write attempt is rejected with an error message.

B.2.1.1.3 READ Function

The READ function is used to read data from the MDS which is located in the antenna field of the SLG U92. It is an "untargeted" command since the MDS is not identified by the ID number.

Byte	0	1	2	3	4	5
Parameter	05	02	00	Address		Length

address	Binary value	0 to maximum length of user data minus 1. Start address of the data to be read from the MDS. The address plus data length must be less than the end address.		
		- 16	=	FFF0 hex Start address of the OTP memory
length	Binary value	1 to 248	=	Length of the user data to be read. Start address + length must be less than the value of the memory length of the MDS in bytes minus 1.
		16	=	Length of the OTP memory

The READ command may only be used for the SLG U92 and then only when no command is queued on the SLG U92. The antenna must be on. Otherwise an error message is generated.
When more than one MDS is located in zone 1, the command is terminated with an error.

When no MDS is located in zone 1, the command waits until an MDS enters zone 1 or a RESET command arrives.

Notice

The 128 bits of user information in the OTP memory are addressed with the start address –16 (FFF0 hex). The 128 bits of user information are written to the MDS with the WRITE command. With the READ command, all 128 bits of information must be requested at one time.

B.2.1.2 System Functions

B.2.1.2.1 RESET Function

The RESET command is used to reset the SLG U92 to a defined state. You determine system behavior of the SLG U92 by specifying the necessary parameters.

Standard settings:

- Read/write distance up to 1.5 m
- Bunch/multitag = 1
- Without proximity switches. The SLG U92 does not evaluate the digital inputs on the service interface.

Byte	0	1	2	3	4	5
Parameter	05	00	00	Standby	Param.	00

⇓

Bit	7	6	5	4	3	2	1	0

Standby	Binary value	Time in which the MDS is to remain in standby after an executed MDS command. This means that, during this time, the MDS remains "awake" so that the next command which must arrive within this standby time can be processed without delay. The value doesn't specify the time directly. Instead it gives a 7 msec factor. For example, a value of 10 means 10 x 7 msec = 70 msec.			
		0	=	No standby time. After each communication with the MDS, the MDS "goes to sleep" again.	
		1 to 200	=	7 msec to 1400 msec	
Param.	Bit pattern	Parameter			
		Bit 7 to 6	=	0	
		Bit 5	=	0	Operation without presence check
			=	1	Operation with presence check (see ANW-MELD acknowledgment telegram)
		Bit 4	=	0	In reserve
		Bit 3 to 0	=	5 hex	Operation mode MOBY U ⇒ MOBY I-commands without expansions

The RESET command may always be sent to the SLG U92 and is executed immediately. If another command is waiting, it is terminated.
After execution of the RESET command, the antenna of the SLG U92 is turned on.

B.2.1.2.2 SLG Status Function (SLG Status/Diagnosis)

This function is used to poll the status of the SLG U92 or to read the diagnostic data from the SLG U92.

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

Mode	Binary value	01 hex	=	SLG status
		02 hex	=	SLG diagnosis I: Request last n function calls
		03 hex	=	SLG diagnosis II: Request last n error messages
		04 hex	=	SLG diagnosis III: Request last n identified MDSs

The SLG-STATUS command may always be sent to the SLG U92 and is executed immediately. When a command such as READ, WRITE or INIT is queued on the SLG U92, it is retained.

B.2.1.2.3 L-UEB Function

This function is used to monitor the connection to the SLG U92 at the logical level.

Byte	0	1	2
Parameter	02	FF	00

The L-UEB command may always be sent to the SLG U92 and is answered immediately. If no return message is received, the connection to the SLG U92 is disconnected (due to a malfunction). When a command such as READ, WRITE or INIT is queued on the SLG U92, it is retained.

B.2.2 Acknowledgments/Messages from the SLG U92

Telegram overview

	Telegram Header			**User Data (Max. of 251 Bytes)**		
Byte	0	1	2	3 to max. of 253 bytes		
Acknowledg-ment/message	AB [hex]	Com-mand [hex]	Status [hex]	User data [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	Status information	
SLG-STATUS (Diagnosis I)	ABL	04	00	S info	Diagnostic information	
SLG-STATUS (Diagnosis II)	ABL	04	00	S info	Diagnostic information	
SLG-STATUS (Diagnosis III)	ABL	04	00	S info	Diagnostic information	
L-UEB	02	FF	05			
Startup	02	00	0F			
ANW-MELD	04	0F	00	00	ANW-S	

AB	=	Telegram length in bytes without the AB byte
ABL	=	Variable telegram length in bytes without the AB byte, depends on the variable user data length

B.2.2.1 Acknowledgments to MDS Functions

B.2.2.1.1 INIT Acknowledgment

Byte	0	1	2
Parameter	02	03	Status

Status	Bit pattern	For status, see appendix B.6.

B.2.2.1.2 WRITE Acknowledgment

Byte	0	1	2
Parameter	02	01	Status

Status	Bit pattern	For status, see appendix B.6.

B.2.2.1.3 READ Acknowledgment

Acknowledgment without error (status = 00 hex)

Byte	0	1	2	3 4	5	6 to max. of 253
Parameter	ABL	02	Status	Address	Length	Data

ABL	Binary value	1 to 253 = Telegram length in bytes without AB byte
Status	Bit pattern	00 hex
Address	Binary value	0 to value: Memory length in bytes minus 1 -16 (FFF0 hex) after OTP memory is read
Length	Binary value	1 to 248 = Length of user data read
Data	Binary info	User data read by MDS

Acknowledgment with error (status not equal 00 hex)

Byte	0	1	2	3 4	5
Parameter	05	02	Status	Address	Length

Status	Bit pattern	For status, see appendix B.6.
Address	Binary value	Start address on the MDS as specified in the function call
Length	Binary value	Length of the data to be read from the MDS as specified in the function call

B.2.2.2 Acknowledgments to System Functions

B.2.2.2.1 RESET Acknowledgment

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

⇓

Byte	VersH	VersL

Status	Bit pattern	For status, see appendix B.6.		
VersH	Binary value	00 hex to FF hex	=	Firmware status (high)
VersL	Binary value	00 hex to FF hex	=	Firmware status (low)
		Example: 01 (high) and 0A (low) = version 1.10		

B.2.2.2.2 SLG-STATUS Acknowledgment (SLG Status)

Acknowledgment without error (status = 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	Bit pattern	00 hex		
S info	Binary value	01 hex	=	SLG status mode
HW	ASCII	HW version		
HW-V	Binary value	HW version		
		00 hex to FF hex	=	Version (high byte)
		00 hex to FF hex	=	Version (low byte)
Url-V	Binary value	Bootstrap loader version		
		00 hex to FF hex	=	Version (high byte)
		00 hex to FF hex	=	Version (low byte)
FW	ASCII format	FW version		
FW-V	Binary value	FW version		
		00 hex to FF hex	=	Version (high byte)
		00 hex to FF hex	=	Version (low byte)

TR	ASCII format	Driver version ‘1’ = 3964R
TR-V	Binary value	Driver version 00 hex to FF hex = Version (high byte) 00 hex to FF hex = Version (low byte)
SS	Binary value	RS 232 / RS 422 01 hex = RS 422 02 hex = RS 232
Baud	Binary value	Baud rate 01 hex = 19.2 Kbaud 02 hex = 38.4 Kbaud 03 hex = 57.6 Kbaud 05 hex = 115.2 Kbaud
dili	Binary value	Distance limit 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	Binary value	Number of MDSs in the antenna field which can be processed (multitag / bunch) = 1
fcon	Binary value	Proximity switch (field ON control) 00 hex = Mode 1: No prox. switches 01 hex = Mode 2: 1 or 2 prox. switches The proximity switches are OR-linked. The field is on while the 1st and/or 2nd proximity switch is on. Otherwise off. 02 hex = Mode 3: 1 or 2 prox. switches The 1st proximity switch turns the field on and the 2nd turns the field off. When two proximity switches exist and one proximity switch is parameterized, the field is automatically turned off when the 2nd proximity switch does not switch within this proximity switch time. If the 2nd proximity switch does not exist, one proximity switch time must be parameterized. After this time, the field is automatically turned off.
ftim	Binary value	Proximity switch time (field ON time) 0 = No proximity time (see proximity switch mode) 1 to 255 = 1 to 255 seconds
Sema	Binary value	Semaphore control (synchronization with SLG) 01 hex = Yes 02 hex = No

ANT	Binary value	Status of antenna		
		01 hex	=	Antenna on
		02 hex	=	Antenna off
Standby	Binary value	Time after an executed MDS command for standby of MDS		
		0	=	No standby. After each time communication with the MDS occurs, the MDS "goes to sleep." The MDS cannot be read or written until after the "sleep time."
		1 to 200	=	7 msec to 1400 msec
ANW	Binary value	Presence (see RESET)		
		00 hex	=	Operation without presence check
		01 hex	=	Operation with presence check (see ANW-MELD acknowledgment)

Acknowledgment with error (status not equal 00 hex)

Byte	0	1	2
Parameter	02	04	Status

Status	Bit pattern	For status, see appendix B.6.

B.2.2.2.3 SLG-STATUS Acknowledgment (SLG Diagnosis I)

Acknowledgment without error (status = 00 hex)

Byte	0	1	2	3	4	5 to 4 + 7	....	to 4 + 7 * n
Parameter	ABL	04	Status	S info	n	1st FKT (n = 1)	....	nth FKT (n = max.)

ABL	Binary value	Telegram length in bytes without AB byte 4 + 7 * n $0 \leq n \leq 33$ 4 to max. of 236
Status	Bit pattern	00 hex
S info	Binary value	02 hex = SLG diagnosis I mode
n	Binary value	Number of functions called last 0 to 33
1st FKT	Binary value	1st function: Function data with length of 7 bytes
„	„	„
nth FKT	Binary value	nth function: Function data with length of 7 bytes

Acknowledgment with error (status not equal 00 hex)

Byte	0	1	2
Parameter	02	04	Status

Status	Bit pattern	For status, see appendix B.6.

B.2.2.2.4 SLG-STATUS Acknowledgment (SLG Diagnosis II)

Acknowledgment without error (status = 00 hex)

Byte	0	1	2	3	4	4 + 1	....
Parameter	ABL	04	Status	S info	n	1st FM (n = 1)	....

4 + n
nth FM (n = max.)

ABL	Binary value	Telegram length in bytes without AB byte 4 + n $0 \leq n \leq 233$ 4 to max. of 238
Status	Bit pattern	00 hex
S info	Binary value	03 hex = SLG diagnosis II mode
n	Binary value	Number of error messages that occurred last 0 to 233
1st FM	Binary value	1st error message (number) – 1 byte
„	„	„
nth FM	Binary value	nth error message (number) – 1 byte

Acknowledgment with error (status not equal 00 hex)

Byte	0	1	2
Parameter	02	04	Status

Status	Bit pattern	For status, see appendix B.6.

B.2.2.2.5 SLG-STATUS Acknowledgment (SLG Diagnosis III)

Acknowledgment without error (status = 00 hex)

Byte	0	1	2	3	4	5 to 4 + 4	....
Parameter	ABL	04	Status	S info	n	1st MDS no. (n = 1)	....

	to 4 + 4 * n
	nth MDS no. (n = max.)

ABL	Binary value	Telegram length in bytes without AB byte 4 + 4 * n $0 \leq n \leq 24$ 4 to max. of 100
Status	Bit pattern	00 hex
S info	Binary value	04 hex = SLG diagnosis III mode
n	Binary value	Number of MDSs identified last 0 to 24
1st MDS no.	Binary value	1st MDS number (4 bytes)
„	„	„
nth MDS no.	Binary value	nth MDS number (4 bytes)

Acknowledgment with error (status not equal 00 hex)

Byte	0	1	2
Parameter	02	04	Status

Status	Bit pattern	For status, see appendix B.6.

B.2.2.2.6 L-UEB Acknowledgment

Byte	0	1	2
Parameter	02	FF	05

B.2.2.3 Messages

B.2.2.3.1 Startup Message

Byte	0	1	2
Parameter	02	00	0F

The SLG U92 sends a startup telegram after the SLG U92 is powered up.

B.2.2.3.2 ANW-MELD Message

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

Status	Bit pattern	For status, see appendix B.6
ANW-S	Binary value	Presence status = Number of MDSs in the field (zone 1) 0 to 12

When the "presence check" bit is set in the RESET telegram, the SLG U92 sends a telegram with the number of MDSs in the field each time "presence" changes in the field (zone 1). When one MDS leaves the field at the same time as another MDS is entering the field, 2 ANW-MELD messages are sent. If several MDSs enter the field simultaneously, each MDS generates an ANW-MELD message. The same applies when the field is exited.

The presence message is sent asynchronously by the SLG U92.

B.3 MOBY I Call-Compatible (Version 2)

B.3.1 Telegrams to the SLG U92

MDS functions

• INIT	Initialize MDS
• WRITE	Write data block
• READ	Read data block
• MDS-STATUS	MDS status/diagnosis

System functions

• RESET	Reset SLG
• SLG-STATUS	SLG status/diagnosis
• SET-ANT	Turn antenna on/off
• END	Conclude communication with MDS
• REPEAT	Repeat last command
• L-UEB	Line monitoring

The RESET command is used to reset the SLG U92 to a defined state. You determine the reaction of the SLG U92 by setting the applicable parameters.

Telegram overview

	Telegram Header			User Data (Max. of 251 Bytes)						
Byte	0	1	2	3 to max. of 253 bytes						
Telegram (Function)	AB [hex]	Com-mand [hex]	Status [hex]	User data [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	=	Telegram length in bytes without the AB byte
ABL	=	Variable telegram length in bytes without the AB byte, depending on the length parameter ⇒ 5 + length

Notice

The data in the telegram overview and the following individual presentations are shown in hexadecimal (hex) format.

B.3.1.1 MDS Functions

The MDS functions INIT, WRITE and READ are used to read or write data from/to the MDS. The MDS function MDS-STATUS is used to poll the status and diagnostic data of the MDS.

B.3.1.1.1 INIT Function

See appendix B.2.1.1.1.

B.3.1.1.2 WRITE Function

See appendix B.2.1.1.2.

B.3.1.1.3 READ Function

See appendix B.2.1.1.3.

B.3.1.1.4 MDS-STATUS Function

This function is used to obtain the status and diagnostic data of the MDS that is located in the antenna field of the SLG U92. It is an "untargeted" call since the MDS is not identified by the ID number.

Byte	0	1	2	3	4	5
Parameter	05	0B	00	Mode	Cweek	Year

Mode	Binary value	00 hex	=	Request status and diagnostic data of an MDS
Cweek	Binary value	1 to 53	=	Current calendar week
Year	Binary value	1 to 99	=	Current year (starting with 1 for 2001)

The MDS-STATUS command may only be sent to the SLG U92 when no command is queued on the SLG U92.

The antenna must be on. Otherwise an error message is generated.
If there is no MDS in zone 1, an error message is sent.
When more than one MDS is located in zone 1, the function is terminated with an error.
The function sequence depends on the fields "cweek" and "year."

a) If the values in both fields are within the value range, the remaining battery life is output in the response.

b) If one of the values is not within the specified value range, remaining battery life cannot be calculated and the function is terminated with an error.

c) If both values contain FF hex days, remaining battery life is not calculated and battery life is specified in the acknowledgment as FFFF hex.

B.3.1.2 System Functions

B.3.1.2.1 RESET Function

The RESET command is used to reset the SLG U92 to a defined state. You determine the system reaction of the SLG U92 by specifying the appropriate parameters.

Standard setting:

- Bunch/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	Binary value	Standby time which the MDS is to wait after an executed MDS command. This means that the MDS remains “awake” during this time so that it will be able to process the next command which must arrive within this standby time without delay. The value does not specify the time directly. Instead it gives a factor of 7 msec (e.g., the value 10 is 10 x 7 msec = 70 msec).

0	=	No standby. The MDS “goes to sleep” after each communication with the MDS.
1 to 200	=	7 msec to 1400 msec

Param.	Bit pattern	Parameter			
		Bit 7 to 6	=	0	
		Bit 5	=	0	No presence check
			=	1	Presence check (see ANW-MELD acknowledgment)
		Bit 4	=	0	In reserve
		Bit 3 to 0	=	5 hex	Mode MOBY U ⇒ MOBY I commands with expansions

dili — Binary value — Distance limit (zone 1)
The read/write range of the SLG U92 (0.5 to 3 m) can be limited in increments of 0.5 m. With the maximum distance of 3 m, 3.5 m must the parameterized as the limit.

Together with the range limitation, the sending capacity can be reduced. For reasons, see the MOBY U manual for configuration, installation and service.

Normal sending capacity			Reduced sending capacity		
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 — Binary value — Proximity switch mode

00 hex	=	Mode 1: Without proximity switches or SLG synchronization
01 hex	=	Mode 2: One or two prox. switches The proximity switches are logically OR-linked. While the 1st and/or the 2nd proximity switch is on, the field is on. Otherwise it is off.
02 hex	=	Mode 3: One of two prox. switches The 1st proximity switch turns the field on and the 2nd proximity switch turns the field off. If there are two proximity switches and a proximity switch time is parameterized, the field is automatically turned off if the 2nd proximity switch does not activate within this proximity switch time. If the 2nd proximity switch is not present, a proximity switch time must be parameterized. The field is automatically turned off after this time expires.
03 hex	=	Mode 4: SLG synchronization (see MOBY U manual for configuration, installation and service).

ftim — Binary value

0	=	Proximity switch time = 0 (when proximity switch mode = 0)
1 to 255	=	Proximity switch time = 1 to 255 seconds

The RESET command may be sent to the SLG U92 at all times. It is executed immediately. Any other queued command is terminated.
After the RESET command has been executed, the antenna of the SLG U92 is on.

B.3.1.2.2 SLG-STATUS Function (SLG Status/Diagnosis)

See appendix B.2.1.2.2.

B.3.1.2.3 SET-ANT Function

This function is used to turn the antenna of the read/write device (SLG U92) on or off.

Byte	0	1	2	3
Parameter	03	0A	00	Mode

Mode	Binary value	01 hex	= Turn on antenna
		02 hex	= Turn off antenna

The SET-ANT command may only be sent to the SLG U92 when no command is queued on the SLG U92 yet.
At the time the antenna is turned on, an MDS may already be present in the field of the SLG U92.
If an MDS is in the field of the SLG U92 when the antenna is turned off, this is reported as not present if the presence check is being used.

B.3.1.2.4 END Function

This function is used to deactivate the standby time (parameterized in RESET telegram) of the MDS which was last processed and is still in the field of the SLG U92. This is done to reduce the current consumption of the MDS.

Byte	0	1	2	3
Parameter	03	08	00	Mode

Mode	Binary value	00 hex	=	Processing with the MDS is concluded. The MDS will leave the field of the SLG U92 (zone 1). No further communication is to occur with this MDS. The parameterized standby time becomes inactive. The SLG U92 removes the MDS from the processing list but continues to keep the MDS in the presence list until the MDS leaves zone 1.
		01 hex	=	Pause in processing with the MDS. The MDS doesn't leave the field of the SLG U92 (zone 1) yet. At least one further communication with the MDS is planned. The parameterized standby time becomes inactive. The SLG U92 continues to keep the MDS in both the processing list and the presence list (e.g., READ command, pause and then WRITE command).

The END command may only be sent to the SLG U92 after a READ, WRITE or INIT command. No command may be waiting on the SLG U92. The antenna must be on. Otherwise an error message is generated.

This command refers to the last processed MDS.
If the MDS has already left zone 1 and mode 01 was selected, an error message is sent. Similarly, if another MDS enters zone 1 and mode 01 was selected, an error message is also created.

B.3.1.2.5 REPEAT Function

This function is used to automatically repeat an MDS command (MDS function) or a command chain (MDS functions) as soon as an MDS enters the antenna field.

• MDS command:	INIT, WRITE, READ and MDS-STATUS The END command cannot be automatically repeated.
• Command chain:	Chain of MDS commands INIT, WRITE, READ and MDS-STATUS and the END command The END command may only be located at the end of the command chain.

This function repeats the MDS command transferred or executed last or the command chain transferred or executed last.

Byte	0	1	2	3
Parameter	03	0D	00	Mode

Mode	Binary value	00 hex	=	Repeat until this command arrives with mode = 1
		01 hex	=	Stop repeating. A started command will be processed until the end.

The MDS command to be executed or the command chain to be executed must contain correct parameters or have already been executed once without errors.

If the MDS command or the command chain is to be used on different types of MDSs (2-kbyte or 32-kbyte), the area to be addressed must be adhered to since otherwise an 0D hex error might occur.

Notice

When the REPEAT function is triggered after a RESET, SLG-STATUS, SET-ANT or END command, the function is rejected with an error status.

When automatic command repetition is activated and an SLG-STATUS is called, the SLG-STATUS is executed asynchronously. Automatic command repetition remains active.

When an additional MDS enters the antenna field while the command is being executed, command execution is terminated with error 1D hex. This means that, with a command chain, every telegram from this time on is acknowledged with an error status. When only one MDS is still located in the field, the command or the command chain is executed on this MDS.

When the MDS command or the command chain without the END command was executed on an MDS located in the antenna field and then another MDS enters the field, the MDS command or command chain to be executed is terminated with error 1D hex. This means that, with a command chain, each telegram is acknowledged with an error status.

Caution

No check is made to determine whether OTP memory was addressed in a write command. With automatic repetition, each MDS would receive the same OTP memory content.

B.3.1.2.6 L-UEB Function

See appendix B.2.1.2.3.

B.3.2 Acknowledgments/Messages from the SLG U92

Telegram overview

	Telegram Header			**User Data (Max. of 251 Bytes)**			
Byte	0	1	2	3 to max. of 253			
Acknowledg-ment/message	AB [hex]	Com-mand [hex]	Status [hex]	User data [hex]			
INIT	02	03	00				
WRITE	02	01	00				
READ	ABL	02	00	Address	Length	Data	
MDS-STATUS	12	0B	00	MDS no.	MDS type	Sum of subframe accesses	
				Sum of search mode accesses	Σ MCOD	Remain. batt.	ST
RESET	05	00	00	FW	00		
SLG-STATUS (SLG status)	1B	04	00	S info	Status information		
SLG-STATUS (diagnosis I)	ABL	04	00	S info	Diagnostic information		
SLG-STATUS (diagnosis II)	ABL	04	00	S info	Diagnostic information		
SLG-STATUS (diagnosis III)	ABL	04	00	S info	Diagnostic information		
SET-ANT	02	0A	00				
END	02	08	00				
REPEAT	02	0D	00				
L-UEB	02	FF	05				
Startup	02	00	0F				
ANW-MELD	04	0F	00	00	ANW-S		

AB	=	Telegram length in bytes without AB byte
ABL	=	Variable telegram length in bytes without AB byte, depending on the variable length of the user data

B.3.2.1 Acknowledgments to MDS Functions

B.3.2.1.1 INIT Acknowledgment

See appendix B.2.2.1.1.

B.3.2.1.2 WRITE Acknowledgment

See appendix B.2.2.1.2.

B.3.2.1.3 READ Acknowledgment

See appendix B.2.2.1.3.

B.3.2.1.4 MDS-STATUS Acknowledgment

Acknowledgment without error (status = 00 hex)

Byte	0	1	2	3 to 6	7	8 to 11	12 13
Parameter	12	0B	Status	MDS no.	MDS type	Sum of subframe accesses	Sum of search mode accesses

14 15	16 17	18
MCOD	Remain. batt.	ST

Status	Bit pattern	00 hex		
MDS no.	Binary value	Value of 2^0 to 2^{31}		
MDS type	Binary value	84 Hex	=	MDS with 2 Kbytes with ECC
		86 Hex	=	MDS with 32 Kbytes with ECC
Sum of subframe accesses	Binary value	Sum of subframe accesses, 32 bits		
Sum of search mode accesses	Binary value	Sum of search mode accesses		
		16 bits	=	Upper 16 bits of a 32-bit value indicating the number of search accesses before the sleep time changed last
MCOD	Binary value	Date of last time sleep-time was changed		
		16 bits	=	byte 14: Calendar week
			=	byte 15: Calendar year (without century)
		If the sleep-time has not changed, calendar week 01 and calendar year 01 are output.		

Remain. batt.	Binary value	16 bits	=	Remaining battery life as percentage
ST	Binary value	Sleep-time: Value set on MDS		
		0	=	20 msec +/- 6.7 msec
		1	=	40 msec +/- 13.3 msec
		2	=	80 msec +/- 26.7 msec
		3	=	160 msec +/- 53.3 msec
		4	=	320 msec +/- 106.7 msec (default)
		5	=	640 msec +/- 213.3 msec
		6	=	1280 msec +/- 426.7 msec
		7	=	2560 msec +/- 853.3 msec
		Statistically, tolerance range of each MDS is equally distributed.		

Acknowledgment with error (status not equal 00 hex)

Byte	0	1	2
Parameter	02	0B	Status

Status	Bit pattern	For status, see appendix B.6.

B.3.2.2 Acknowledgments to System Functions

B.3.2.2.1 RESET Acknowledgment

See appendix B.2.2.2.1.

B.3.2.2.2 SLG-STATUS Acknowledgment (SLG Status)

Acknowledgment without error (status = 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	Bit pattern	00 hex

S info	Binary value	01 hex	=	SLG status mode			
HW	ASCII	HW version					
HW-V	Binary value	HW version					
		00 hex to FF hex	=	Version (high byte)			
		00 hex to FF hex	=	Version (low byte)			
Url-V	Binary value	Bootstrap loader version					
		00 hex to FF hex	=	Version (high byte)			
		00 hex to FF hex	=	Version (low byte)			
FW	ASCII format	FW version					
FW-V	Binary value	FW version					
		00 hex to FF hex	=	Version (high byte)			
		00 hex to FF hex	=	Version (low byte)			
TR	ASCII format	Driver version ‘1’ = 3964R					
TR-V	Binary value	Driver version					
		00 hex to FF hex	=	Version (high byte)			
		00 hex to FF hex	=	Version (low byte)			
SS	Binary value	RS 232 / RS 422					
		01 hex	=	RS 422			
		02 hex	=	RS 232			
Baud	Binary value	Baud rate					
		01 hex	=	19.2 Kbaud			
		02 hex	=	38.4 Kbaud			
		03 hex	=	57.6 Kbaud			
		05 hex	=	115.2 Kbaud			
dili	Binary value	Distance limit					
		Normal sending capacity			Reduced sending capacity		
		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	Binary value	Number of MDSs in the antenna field which can be processed (multitag / bunch) = 1					
fcon	Binary value	Proximity switch (field ON control)					
		00 hex	=	Mode 1: Without proximity switches or SLG synchronization			
		01 hex	=	Mode 2: 1 or 2 prox. switches The proximity switches are OR-linked. The field is on while the 1st and/or 2nd proximity switch is on. Otherwise off.			
		02 hex	=	Mode 3: 1 or 2 prox. switches The 1st proximity switch turns the field on and the 2nd turns the field off.			

		03 hex	=	Mode 4: SLG synchronization (see MOBY U manual for configuration, installation and service).
		When two proximity switches exist and one proximity switch is parameterized, the field is automatically turned off when the 2nd proximity switch does not switch within this proximity switch time. If the 2nd proximity switch does not exist, one proximity switch time must be parameterized. After this time, the field is automatically turned off.		
ftim	Binary value	Proximity switch time (field ON time)		
		0	=	No proximity time (see proximity switch mode)
		1 to 255	=	1 to 255 seconds
Sema	Binary value	Semaphore control (synchronization with SLG)		
		01 hex	=	Yes
		02 hex	=	No

ANT	Binary value	Status of antenna		
		01 hex	=	Antenna on
		02 hex	=	Antenna off
Standby	Binary value	Time after an executed MDS command for standby of MDS		
		0	=	No standby. After each time communication with the MDS occurs, the MDS "goes to sleep." The MDS cannot be read or written until after the "sleep time."
		1 to 200	=	7 msec to 1400 msec
ANW	Binary value	Presence (see RESET)		
		00 hex	=	Operation without presence check
		01 hex	=	Operation with presence check (see ANW-MELD acknowledgment)

Acknowledgment with error (status not equal 00 hex)

Byte	0	1	2
Parameter	02	04	Status

Status	Bit pattern	For status, see appendix B.6.

B.3.2.2.3 SLG-STATUS Acknowledgment (SLG Diagnosis I)

See appendix B.2.2.2.3.

B.3.2.2.4 SLG-STATUS Acknowledgment (SLG Diagnosis II)

See appendix B.2.2.2.4.

B.3.2.2.5 SLG-STATUS Acknowledgment (SLG Diagnosis III)

See appendix B.2.2.2.5.

B.3.2.2.6 SET-ANT Acknowledgment

Byte	0	1	2
Parameter	02	0A	Status

Status	Bit pattern	For status, see appendix B.6.

B.3.2.2.7 END Acknowledgment

Byte	0	1	2
Parameter	02	08	Status

Status	Bit pattern	For status, see appendix B.6.

B.3.2.2.8 REPEAT Acknowledgment

Byte	0	1	2
Parameter	02	0D	Status

Status	Bit pattern	For status, see appendix B.6

B.3.2.2.9 L-UEB Acknowledgment

See appendix B.2.2.2.6.

B.3.2.3 Messages

B.3.2.3.1 Startup Message

See appendix B.2.2.3.1.

B.3.2.3.2 ANW-MELD Message

See appendix B.2.2.3.2.

B.4 MOBY U with Multitag Processing (Version 3)

B.4.1 Telegrams to the SLG U92

MDS functions

• INIT	Initialize MDS
• WRITE	Write data block
• READ	Read data block
• GET	Get MDS
• COPY	Copy data from MDS 1 to MDS 2
• MDS-STATUS	MDS status/diagnosis

System functions

• RESET	Reset SLG
• SLG-STATUS	SLG status/diagnosis
• SET-ANT	Turn antenna on/off
• END	End communication with MDS
• REPEAT	Repeat last command
• L-UEB	Line monitoring

You can use the RESET command to reset the SLG U92 to a defined state. By setting the parameters you can specify the system behavior of the SLG U92.

Telegram overview

	Telegram Header			User Data (Max. of 251 Bytes)						
Byte	0	1	2	3 to max. of 253						
Telegram (Function)	AB [hex]	Com-mand [hex]	Status [hex]	User data [hex]						
INIT	0A	03	00	MDS no.	date	00	length			
WRITE	ABL	01	00	MDS no.	address	length	data			
READ	09	02	00	MDS no.	address	length				
GET	06	0C	00	mode	address	length				
COPY	10	07	00	MDS no. 1	address1	length	MDS no. 2		address2	
MDS-STATUS	09	0B	00	MDS no.	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 no.	mode					
REPEAT	03	0D	00	mode						
L-UEB	02	FF	00							

AB	=	Telegram length in bytes without the AB byte
ABL	=	Variable telegram length in bytes without the AB byte, depending on the length parameter ⇒ 5 + length

Notice

The data in the telegram overview and the following individual presentations are shown in hexadecimal (hex) format.

B.4.1.1 MDS Functions

- You can use the MDS functions INIT, READ and WRITE to read or write data to and from the MDS.
- You can use the GET function to find out which MDS is located in the field. You can also read data from this MDS at the same time.
- You can use the COPY function to copy data. You can copy one data area or the entire contents of a data carrier from one MDS to another.
- You can use the MDS-STATUS function to query status and diagnostic data from an MDS.

B.4.1.1.1 INIT Function

You can use the INIT function to "non-specifically" or "specifically" initialize with a bit pattern an MDS that is located in the antenna field of the SLG U92.

- An initialization call is "non-specific" when the call does not include the identification number of the MDS. Only one MDS may be located in the antenna field.
- An initialization call is "specific" when the call includes the identification number of the MDS. More than one MDS may be located in the antenna field. You can determine the identification number of the MDS with the GET function.

Byte	0	1	2	3 to 6	7	8	9 10
Parameter	0A	03	00	MDS no.	date	00	length

MDS no.	Binary value	0	=	When an MDS is located in the antenna field and a "non-specific" initialization call is to be executed
		Value 2^0 to 2^{32} -1	=	MDS no. of the MDS that is to be initialized
Date	Binary value	Bit pattern (00 hex to FF hex) with which the data carrier is to be initialized (written)		
Length	Binary value	32768	=	Length in bytes of the data memories MDS U524, MDS U525 and MDS U589
		2048	=	Length in bytes of data memory MDS U313 and MDS U315

The INIT command may only be used on the SLG U92 when no command is yet queued on the SLG U92. The antenna must be on. Otherwise an error message is generated.
If the call is "non-specific" and there is more than one MDS in zone 1, the command is terminated with an error.
If the call is "non-specific" and there is no MDS in zone 1, the command waits until an MDS moves into zone 1 or the RESET command arrives.
If the call is "specific" and the MDS with the specified MDS no. is not located in zone 1, the command is terminated with an error.

B.4.1.1.2 WRITE Function

You can use the WRITE function to "non-specifically" or "specifically" write data to an MDS that is located in the antenna field of the SLG U92.

- A write call is "non-specific" when the call does not include the identification number of the MDS. Only one MDS may be located in the antenna field.
- A write call is "specific" when the call includes the identification number of the MDS. More than one MDS may be located in the antenna field. You can determine the identification number of the MDS with the GET function.

Byte	0	1	2	3 to 6	7 8	9	10 to max. of 253
Parameter	ABL	01	00	MDS no.	address	length	data

ABL	Binary value	1 to 253	=	Telegram length in bytes without the AB byte
MDS no.	Binary value	0	=	When an MDS is located in the antenna field and a “non-specific” write call is to be performed
		Value 2^0 to 2^{32} -1	=	MDS no. of the MDS which is to be written
Address	Binary value	0 to maximum length of user data minus 1 Start address on the MDS of the data to be written The address plus data length must be less than the end address..		
		- 16	=	FFF0 hex Start address of the OTP memory
Length	Binary value	1 to 244	=	Length of the user data to be written
		16	=	Length of the OTP memory
Data	Binary information	User data to be written to the MDS		

The WRITE command may only be used on the SLG U92 when no command is yet queued on the SLG U92. The antenna must be on. Otherwise an error message is generated.
If the call is "non-specific" and there is no MDS in zone 1, the command waits until an MDS moves into zone 1 or the RESET command arrives.
If the call is "non-specific" and there is more than one MDS in zone 1, the command is terminated with an error.
If the call is "specific" and the MDS with the specified MDS no. is not located in zone 1, the command is terminated with an error.

Notice

The 128 bits of user information in the OTP memory are addressed with the start address –16 (FFF0 hex). The OTP memory can only be written once. The write call must contain all 128 bits of information. A second attempt to write is rejected with an error message.

B.4.1.1.3 READ Function

You can use the READ function to "non-specifically" or "specifically" read data from an MDS that is located in the antenna field of the SLG U92.

- A read call is "non-specific" when the call does not include the identification number of the MDS. Only one MDS may be located in the antenna field.
- A read call is "specific" when the call includes the identification number of the MDS. More than one MDS may be located in the antenna field. You can determine the identification number of the MDS with the GET function.

Byte	0	1	2	3 bis 6	7	8	9
Parameter	09	02	00	MDS no.	address		length

MDS no.	Binary value	0	=	When an MDS is located in the antenna field and a "non-specific" read call is to be performed
		Value 2^0 to 2^{32} -1	=	MDS no. of the MDS which is to be read
Address	Binary value	0 to maximum length of user data minus 1 Start address on the MDS of the data to be read The address plus data length must be less than the end address.		
		- 16	=	FFF0 hex Start address of the OTP memory
Length	Binary value	1 to 244	=	Length of the user data to be read
		16	=	Length of the OTP memory

The READ command may only be used on the SLG U92 when no command is yet queued on the SLG U92. The antenna must be on. Otherwise an error message is generated.
If the call is "non-specific" and there is more than one MDS in zone 1, the command is terminated with an error.
If the call is "non-specific" and there is no MDS in zone 1, the command waits until an MDS moves into zone 1 or the RESET command arrives.
If the call is "specific" and the MDS with the specified MDS no. is not located in zone 1, the command is terminated with an error.

Notice

The 128 bits of user information in the OTP memory are addressed with the start address –16 (FFF0 hex). The 128 bits of user information are written to the MDS with the WRITE command. The read call must request all 128 bits of information.

B.4.1.1.4 GET Function

You can use the GET function to determine which MDS is located in the field. At the same time you can read these data from this MDS.

Byte	0	1	2	3	4 5	6
Parameter	06	0C	00	mode	address	length

Mode	Binary value	0	=	Don't read data from MDS.
		1	=	Read data from MDS.
Address	Binary value	0	=	If no data on MDS
		0 to maximum length of user data minus 1 Start address of the data to be read on the MDS The address plus data length must be less than the end address..		
		- 16	=	FFF0 hex Start address of the OTP memory
Length	Binary value	1 to x	=	Length of the user data to be read x = (242 – (4 * bunch size)) / bunch size
		16	=	Length of the OTP memory

The GET command can only be used on the SLG U92 when no command is yet queued on the SLG U92. The antenna must be on. Otherwise an error message is generated.

If there are more MDSs in zone 1 than are permitted by the "bunch" parameter in the RESET telegram, the number of MDSs are reported without ID numbers in an error message. The reported MDSs can then not be processed.

Notice

The 128 bits of user information in the OTP memory are addressed with start address –16 (FFF0 hex). The 128 bits of user information are written to the MDS with the WRITE command. A GET call must include all 128 bits of user information.

B.4.1.1.5 COPY Function

You can use the COPY function to "specifically" copy the following data from one MDS to another.

- A data area or
- The complete contents of a data carrier

This function can be used to write (i.e., copy) data directly via the user program from one MDS as the source to another MDS as the destination.

Byte	0	1	2	3 bis 6	7 8	9 10	11 bis 14	15 16
Parameter	10	0C	00	MDS no. 1	address1	length	MDS no. 2	address2

MDS no. 1	Binary value	Value 2^0 to 2^{32} - 1	=	MDS no. of MDS 1 which is to be copied
Address1	Binary value	0 to maximum length of user data minus 1 Start address of the data to be copied from MDS 1 The address plus data length must be less than the end address.		
Length	Binary value	Number of bytes to be copied: 1 to maximum length of the data memory		
MDS no. 2	Binary value	Value 2^0 to 2^{32} - 1	=	MDS no. of MDS 2 to which data are to be copied
Address2	Binary value	0 to maximum length of the user data minus 1 Start address on MDS 2 starting at which the data are to be written The address plus data length must be less than the end address.		

The COPY command may only be used on the SLG U92 when no command is yet queued on the SLG U92. The antenna must be on. Otherwise an error message is generated.

If the two specified MDSs are not located in zone 1, the command is terminated with an error.

Notice

The OTP memory cannot be copied with the COPY command.

B.4.1.1.6 MDS-STATUS Function

You can use this function to "non-specifically" or "specifically" determine the status and diagnostic data of an MDS which is located in the antenna field of the SLG U92.

- A read call is "non-specific" when the call does not include the identification number of the MDS. Only one MDS may be located in the antenna field.
- A read call is "specific" when the call includes the identification number of the MDS. More than one MDS may be located in the antenna field. You can determine the identification number of the MDS with the GET function.

Byte	0	1	2	3 to 6	7	8	9
Parameter	09	0B	00	MDS no.	mode	cweek	year

MDS no.	Binary value	0	=	When an MDS is located in the antenna field and a "non-specific" status query is to be performed
		Value 2^0 to 2^{32} -1	=	MDS no. of the MDS which is to be queried
Mode	Binary value	00 hex	=	Request status and diagnostic data of an MDS
Cweek	Binary value	1 to 53	=	Current calendar week
Year	Binary value	1 to 99	=	Current year (beginning with 1 for 2001)

The MDS-STATUS command may only be issued to the SLG U92 when no command is queued on the SLG U92.

The antenna must be on. Otherwise an error message is generated.
When the call is "non-specific" and there is no MDS in zone 1, an error message is generated.
When the call is "non-specific" and there is more than one MDS in zone 1, the command is terminated with an error.
When the call is "specific" and the MDS with the specified MDS no. is not located in zone 1, the command is terminated with an error.

The function sequence depends on the fields "cweek" and "year."

a) If the value in both fields is within the value range, the remaining battery life is output in the response.

b) If one of the values is outside the specified value range, the remaining battery life cannot be calculated and the function is terminated with an error.

c) If both values are "FF hex" days, the remaining battery life cannot be calculated and FFFF hex is given in the acknowledgment as the battery life.

B.4.1.2 System Functions

B.4.1.2.1 RESET Function

You can use the RESET command to reset the SLG U92 to a defined state and specify the system behavior of the SLG U92 by setting the parameters appropriately.

Standard setting:

- Bunch/multitag = 1

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

⇓

Bit	7	6	5	4	3	2	1	0

standby	Binary value	Standby time during which MDS is to assume standby mode after an MDS command is executed. This means that the MDS remains "awake" so that it can process the next command (which must arrive during this standby time) without delay. The value specifies a factor of 7 msec and not the direct time. For instance, the value 10 corresponds to 10 x 7 msec = 70 msec.			
		0	=	No standby mode. The MDS "goes to sleep" again after each communication with the MDS.	
		1 to 200	=	7 msec to 1400 msec	
param	Bit pattern	Parameter			
		Bit 7 to 6	=	0	
		Bit 5	=	0	Operation without presence
			=	1	Operation with presence (see ANW-MELD acknowledgment)
		Bit 4	=	0	In reserve
		Bit 3 to 0	=	6 hex	Operating mode MOBY U ⇒ multitag processing/bunch
dili	Binary value	Range limitation (zone 1): The read/write range of the SLG U92 (0.5 to 3 m) can be limited in increments of 0.5 m. 3.5 m must be parameterized for the maximum distance of 3.5 m.			
		Together with the range limitation, the sending capacity can be reduced. For reasons, see the MOBY U manual for configuration, installation and service.			

Normal sending capacity			Reduced sending capacity		
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	Binary value	Multitag/bunch					
		1 to 12	=	Number of MDSs (multitag/bunch) that can be processed in the antenna field (zone 1).			
fcon	Binary value	Proximity switch mode					
		00 hex	=	Mode 1: Without proximity switches or SLG synchronization			
		01 hex	=	Mode 2: One or two proximity switches The proximity switches are logically OR-linked. While the 1st and/or 2nd proximity switch is/are on, the field is on. Otherwise it is off.			
		02 hex	=	Mode 3: One or two proximity switches The 1st proximity switch turns the field on. The 2nd proximity switch turns the field off. When there are two proximity switches and a proximity switch time is parameterized, the field is automatically turned off when the 2nd proximity switch does not switch within this proximity switch time. When there is no proximity switch, a proximity switch time must be parameterized. After this time the field is automatically turned off.			
		03 hex	=	Mode 4: SLG synchronization (see MOBY U manual for configuration, installation and service).			
ftim	Binary value	0	=	Proximity switch time = 0 (when proximity switch mode = 0)			
		1 to 255	=	Proximity switch time = 1 to 255 seconds			

The RESET command may be used on the SLG U92 at all times. It is executed immediately. If another command is queued, it is terminated.
After the RESET command is executed, the antenna of the SLG U92 is on.

B.4.1.2.2 SLG-STATUS Function (SLG Status/Diagnosis)

See appendix B.3.1.2.2.

B.4.1.2.3 SET-ANT Function

You can use this function to turn the antenna of the read/write device (SLG U92) on or off.

Byte	0	1	2	3
Parameter	03	0A	00	mode

ode	Binary value	01 hex	= Turn on antenna.
		02 hex	= Turn off antenna.

The SET-ANT command may only be used on the SLG U92 when no command is yet queued on the SLG U92.
At the time the antenna is turned on, one or more MDSs may already be located in the field of the SLG U92.
If one or more MDSs are located in the field of the SLG U92 when the antenna is turned off, each individual MDS is reported as "not present" when presence mode is being used.

B.4.1.2.4 END Function

You can use this function to reduce MDS power consumption by deactivating the standby time (parameterized in the RESET telegram) of an MDS located in the antenna field.

Byte	0	1	2	3 to 6	7
Parameter	07	08	00	MDS no.	mode

MDS no.	Binary value	0	=	When an MDS is located in the antenna field and a "non-specific" call is to be performed
		Value 2^0 to 2^{32} -1	=	MDS no. of the MDS for which the standby time is to be deactivated
Mode	Binary value	00 hex	=	Processing with the MDS is finished. The MDS will exit the field of the SLG U92 (zone 1). No more communication is to take place with this MDS. The parameterized standby time is deactivated. The SLG U92 removes the MDS from the processing list but retains the MDS in the presence list until the MDS actually exits zone 1.

01 hex	=	Processing pause with the MDS. The MDS doesn't exit the field of the SLG U92 (zone 1) yet. At least one other communication with the MDS is planned. The parameterized standby time is deactivated. The SLG U92 retains the MDS in the processing list and in the presence list. Example: READ command, pause, and then WRITE command

The END command may only be used on the SLG U92 after the READ, WRITE or INIT command. No command may be queued on the SLG U92. The antenna must be on. Otherwise an error message is generated.

If there is a "non-specific" call and there is more than one MDS in zone 1 or the last processed MDS has left zone 1 and mode 01 was selected, the command is terminated with an error.

When the call is "specific" and the MDS with the specified MDS no. is not located in zone 1, the command is terminated with an error.

B.4.1.2.5 REPEAT Function

This function is used to automatically repeat an MDS command (MDS function) or a command chain (MDS functions) as soon as an MDS enters the antenna field.

• MDS command:	INIT, WRITE, READ and MDS-STATUS The GET, COPY and END commands cannot be automatically repeated.
• Command chain:	Chain of MDS commands INIT, WRITE, READ and MDS-STATUS and the END command The END command may only be located at the end of the command chain.

This function repeats the MDS command last transferred or executed or the command chain last transferred or executed.

With the MDS command(s) to be executed, the MDS no. must be preassigned with zero. No MDS no. may be entered.
The REPEAT function is possible when:

- Bunch = 1 is parameterized and an MDS is located in the antenna field or
- Bunch > 1 is parameterized and only one MDS is located in the antenna field.

Byte	0	1	2	3
Parameter	03	0D	00	mode

Mode	Binary value	00 hex	=	Repeat until this command returns mode = 1
		01 hex	=	Conclude repetition. An already started command is processed until the end.

The MDS command to be executed or the command chain to be executed must contain correct parameters or have already been executed once without errors.

If the MDS command or the command chain is to be used on different types of MDSs (2-kbyte or 32-kbyte), the area to be addressed must be adhered to since otherwise an 0D hex error might occur.
Repeat mode is also retained even after an error occurs.

Notice

When the REPEAT function is triggered after a RESET, SLG-STATUS, SET-ANT, GET, COPY or END command, the function is rejected with an error status.

When automatic command repetition is activated and an SLG-STATUS is called, the SLG-STATUS is executed asynchronously. Automatic command repetition remains active.

When an additional MDS enters the antenna field while the command is being executed, command execution is terminated with error 1D hex. This means that, with a command chain, every telegram from this time on is acknowledged with an error status. When only one MDS is still located in the field, the command or the command chain is executed on this MDS.

When the MDS command or the command chain without the END command was executed on an MDS located in the antenna field and then another MDS enters the field, the MDS command or command chain to be executed is terminated with error 1D hex. This means that, with a command chain, each telegram is acknowledged with an error status.

Caution

No check is made to determine whether OTP memory was addressed in a write command. With automatic repetition, each MDS would receive the same OTP memory content.

B.4.1.2.6 L-UEB Function

See appendix B.2.1.2.3.

B.4.2 Acknowledgments/Messages from the SLG U92

Telegram overview

	Telegram Header			**User Data (Max. of 251 Bytes)**			
Byte	0	1	2	3 to max. of 253			
Acknowledg-ment/message	AB [hex]	Com-mand [hex]	Status [hex]	User data [hex]			
INIT	06	03	00	MDS no.			
WRITE	06	01	00	MDS no.			
READ	ABL	02	00	MDS no.	address	length	data
GET	ABL	0C	00	Number of MDSs	1st MDS no.	…	nth MDS no.
				address	length	data MDS 1	
				…	data MDS n		
COPY	0A	07	00	MDS no. 1	MDS no. 2		
MDS-STATUS	12	0B	00	MDS no.	MDS type	Σ Subframe access	
				Σ Search mode access	Σ MCOD	Remain. batt.	ST
RESET	05	00	00	FW	00		
SLG-STATUS (SLG status)	1B	04	00	S info	Status information		
SLG-STATUS (diagnosis I)	ABL	04	00	S info	Diagnostic information		
SLG-STATUS (diagnosis II)	ABL	04	00	S info	Diagnostic information		
SLG-STATUS (diagnosis III)	ABL	04	00	S info	Diagnostic information		
SET-ANT	02	0A	00				
END	06	08	00	MDS no.			
REPEAT	02	0D	00				
L-UEB	02	FF	05				
Startup	02	00	0F				
ANW-MELD	04	0F	00	00	ANW-S		

AB	=	Telegram length in bytes without the AB byte
ABL	=	Variable telegram length in bytes without the AB byte, depending on the variable length of the user data

B.4.2.1 Acknowledgments to MDS Functions

B.4.2.1.1 INIT Acknowledgment

Acknowledgment without error (status = 00 hex)

Byte	0	1	2	3 to 6
Parameter	06	03	status	MDS no.

Status	Bit pattern	00 hex		
MDS no.	Binary value	Value 2^0 to 2^{32} -1	=	MDS no. of the initialized MDS

Acknowledgment with error (status not equal 00 hex)

Byte	0	1	2	3 to 6
Parameter	06	03	status	MDS no.

Status	Bit pattern	For status, see appendix B.6.		
MDS no.	Binary value	Value 2^0 to 2^{32} -1	=	MDS no. of the not initialized MDS (“specific” initialization call with error)
		0	=	“Non-specific” initialization call with error

B.4.2.1.2 WRITE Acknowledgment

Acknowledgment without error (status = 00 hex)

Byte	0	1	2	3 to 6
Parameter	06	01	status	MDS no.

Status	Bit pattern	00 hex		
MDS no.	Binary value	Value 2^0 to 2^{32} -1	=	MDS no. of the written MDS

Acknowledgment with error (status not equal 00 hex)

Byte	0	1	2	3 to 6
Parameter	06	01	status	MDS no.

Status	Bit pattern	For status, see appendix B.6.		
MDS no.	Binary value	Value 2^0 to 2^{32} -1	=	MDS no. of the not written MDS (“specific” write call with error)
		0	=	“Non-specific” write call with error

B.4.2.1.3 READ Acknowledgment

Acknowledgment without error (status = 00 hex)

Byte	0	1	2	3 to 6	7 8	9	10 to max. of 253
Parameter	ABL	02	status	MDS no.	address	length	data

ABL	Binary value	1 to 253	=	Telegram length in bytes without the AB byte
Status	Bit pattern	00 hex		
MDS no.	Binary value	Value 2^0 to 2^{32} - 1	=	MDS no. of the read MDS
Address	Binary value	0 to value: Memory length in bytes minus 1		
		-16 (FFF0 hex) after reading the OTP memory		
Length	Binary value	1 to 244	=	Length of the read user data
Data	Binary information	User data read from the MDS		

Acknowledgment with error (status not equal 00 hex)

Byte	0	1	2	3 to 6
Parameter	06	01	status	MDS no.

Status	Bit pattern	For status, see appendix B.6.		
MDS no.	Binary value	Value 2^0 to 2^{32} -1	=	MDS no. of the not read MDS (“specific” read call with error)
		0	=	“Non-specific” read call with error

B.4.2.1.4 GET Acknowledgment

Acknowledgment without error (status = 00 hex)

Depending on the function call and whether MDSs are located in the antenna field (zone 1), the acknowledgment has the following structure.

GET function with mode = 0 (without MDS data) and no MDS in the field

Byte	0	1	2	3
Parameter	03	0C	status	Number of MDSs

Status	Bit pattern	00 hex		
Number of MDSs	Binary value	0	=	No MDS in the field

GET function with mode = 0 (without MDS data) and at least 1 MDS in the antenna field (zone 1)

Byte	0	1	2	3	4	to	3 + 4 * n
Parameter	ABL	0C	status	Number of MDSs	1st MDS no.	...	nth MDS no.

ABL	Binary value	Telegram length in bytes without the AB byte		
		7 to 3 + 4 * n		$1 < n \le 12$; n = Number of MDSs in the antenna field (zone 1) 4 bytes for each MDS no.
Status	Bit pattern	00 hex		
Number of MDSs	Binary value	0 to 12	=	Number of MDSs in the antenna field (zone 1) = n
1st MDS no.	Binary value	Value 2^0 to 2^{32} -1	=	MDS no. of the 1st MDS
...				
nth MDS no.	Binary value	Value 2^0 to 2^{32} -1	=	MDS no. of the nth MDS

GET function with mode = 1 (with MDS data) and no MDS in the field

Byte	0	1	2	3
Parameter	03	0C	status	Number of MDSs

Status	Bit pattern	00 hex		
Number of MDSs	Binary value	0	=	No MDS in the field

GET function with mode = 1 (with MDS data) and at least 1 MDS in the antenna field (zone 1)

Byte	0	1	2	3	4	to	3 + 4 * n
Parameter	ABL	0C	status	Number of MDSs	1st MDS no.	...	nth MDS no.

4 + 4 * n to 5 + 4 * n	6 + 4 * n
address	length

7 + 4 * n	to	6 + 4 * n + length * n
data MDS 1	…	data MDS n

ABL	Binary value	Telegram length in bytes without the AB byte		
		14 to 6 + 4 * n + length * n		$1 < n \leq 12$; n = Number of MDSs in the antenna field (zone 1) 4 bytes for each MDS no.
Status	Bit pattern	00 hex		
Number of MDSs	Binary value	0 to 12	=	Number of MDSs in the antenna field (zone 1) = n
1st MDS no.	Binary value	Value 2^0 to 2^{32} -1	=	MDS no. of the 1st MDS
...				
nth MDS no.	Binary value	Value 2^0 to 2^{32} -1	=	MDS no. of the nth MDS
Address	Binary value	See GET function, appendix B.4.1.1.4		
Length	Binary value	See GET function, appendix B.4.1.1.4		
Data MDS 1	Binary information	User data read from 1st MDS		
…				
Data MDS n	Binary information	User data read from nth MDS		

Acknowledgment with error (status not equal 00 hex)

Byte	0	1	2
Parameter	02	0C	status

Status	Bit pattern	For status, see appendix B.6.

B.4.2.1.5 COPY Acknowledgment

Acknowledgment without error (status = 00 hex)

Byte	0	1	2	3 to 6	7 to 10
Parameter	0A	07	status	MDS no. 1	MDS no. 2

Status	Bit pattern	00 hex		
MDS no. 1	Binary value	Value 2^0 to 2^{32} -1	=	MDS no. of the read MDS (source)
MDS no. 2	Binary value	Value 2^0 to 2^{32} -1	=	MDS no. of the written MDS (destination)

Acknowledgment with error (status not equal 00 hex)

Byte	0	1	2	3 to 6	7 to 10
Parameter	0A	07	status	MDS no. 1	MDS no. 2

Status	Bit pattern	For status, see appendix B.6.		
MDS no. 1	Binary value	Value 2^0 to 2^{32} -1	=	MDS no. of the not read MDS (source)
		0	=	MDS no. 1 not processed
MDS no. 2	Binary value	Value 2^0 to 2^{32} -1	=	MDS no. of the not written MDS (destination)
		0	=	MDS no. 2 not processed

B.4.2.1.6 MDS-STATUS Acknowledgment

See appendix B.3.2.1.4.

B.4.2.2 Acknowledgments to System Functions

B.4.2.2.1 RESET Acknowledgment

See appendix B.2.2.2.1.

B.4.2.2.2 SLG-STATUS Acknowledgment (SLG Status)

See appendix B.3.2.2.2.

B.4.2.2.3 SLG-STATUS Acknowledgment (SLG Diagnosis I)

See appendix B.2.2.2.3.

B.4.2.2.4 SLG-STATUS Acknowledgment (SLG Diagnosis II)

See appendix B.2.2.2.4.

B.4.2.2.5 SLG-STATUS Acknowledgment (SLG Diagnosis III)

See appendix B.2.2.2.5.

B.4.2.2.6 SET-ANT Acknowledgment

See appendix B.3.2.2.6.

B.4.2.2.7 END Acknowledgment

Acknowledgment without error (status = 00 hex)

Byte	0	1	2	3 to 6
Parameter	06	08	status	MDS no.

Status	Bit pattern	00 hex		
MDS no.	Binary value	Value 2^0 to 2^{32} -1	=	MDS no.

Acknowledgment with error (status not equal 00 hex)

Byte	0	1	2	3 to 6
Parameter	06	08	status	MDS no.

Status	Bit pattern	For status, see appendix B.6.		
MDS no.	Binary value	Value 2^0 to 2^{32} -1	=	MDS no. (“specific” call with error)
		0	=	MDS no. not processed (“non-specific” call with error)

B.4.2.2.8 REPEAT Acknowledgment

See appendix B.3.2.2.8.

B.4.2.2.9 L-UEB Acknowledgment

See appendix B.2.2.2.6.

B.4.2.3 Messages

B.4.2.3.1 Startup Message

See appendix B.2.2.3.1.

B.4.2.3.2 ANW-MELD Message

See appendix B.2.2.3.2.

B.5 Command Chaining

Command chaining can be used to speed up processing of large and/or different address areas on the MDS.

Normally the SLG U92 stores only one MDS command in its memory and executes this command. This means that a maximum of 248 bytes (without multitagging) or 244 bytes (with multitagging) can be read or written with one command. Another command cannot be issued until the previous command is acknowledged.

To speed up reading or writing more than 248 bytes (without multitagging), 244 bytes (with multitagging) and/or different address areas, several commands can be chained and sent to the SLG U92 and stored on the SLG U92.

For processing large and/or different data areas, different functions can also be chained together.

The commands of the command chain are identified by a bit in the upper 4 bits of the command byte (2nd byte = byte 1). Bit 6 is set to "1" and bits 4, 5 and 7 must be "0." In the last command in the command chain bit 6 must equal "0." This signals the end of the chain. The lower 4 bits of the command byte contain the function ID. This means that all the commands that are chained together must have the command type 4x hex. The last command in a chain must have type 0x hex.

The command chain can already be sent to the SLG U92 before the MDS to be processed is located in the field of the SLG U92. As soon as the MDS enters the field and is detected by the SLG U92, the SLG begins executing the chained commands and returns the acknowledgments with data. If the processing of the command chain can already be started while the command chain is being sent to the SLG U92, the first acknowledgments may arrive during the sending procedure.

Notice

The maximum length of a command chain may not exceed 150 commands.

Possible chaining variations

a) n x WRITE

b) n x READ

c) MDS commands in any order: INIT, WRITE, READ and/or MDS-STATUS

d) MDS commands in any order: a), b) or c) plus END command

e) RESET command, followed by versions a), b), c) or d)

f) SET-ANT (EIN), plus followed by versions a), b), c) or d)

g) Versions a), b), c), d), e) or f) plus SET-ANT (AUS)

h) Version a), b), c), d) or f) plus SET-ANT (AUS) with SLG-STATUS in any position

i) RESET command, plus followed by versions a), b), c) or d) with SLG-STATUS in any position after the RESET

Commands which are not permitted in the command chain or illegal chaining

- COPY, GET and REPEAT are not permitted in the command chain (COPY and GET only with multitagging)
- RESET within or at the end of a command chain
- END at the beginning or within a command chain
- END without preceding MDS command
- SET-ANT (AUS) at the beginning or before the last MDS command or before the END

Example of chained MDS commands

- 3 READ commands, chained (506 bytes in one data block)

AB	Com-mand	Status	Address	Length	1st command
05	42	00	00 00	F8	Read 248 bytes

AB	Com-mand	Status	Address	Length	2nd command
05	42	00	00 F8	F8	Read 248 bytes

AB	Com-mand	Status	Address	Length	3rd command (last command)
05	02	00	01 F0	0A	Read 10 bytes

- 3 READ commands, chained (3 unrelated data blocks)

AB	Com-mand	Status	Address	Length	
05	42	00	00 00	14	1st command Read 20 bytes

AB	Com-mand	Status	Address	Length	
05	42	00	00 F0	1F	2nd command Read 31 bytes

AB	Com-mand	Status	Address	Length	
05	02	00	02 01	A5	3rd command (last command) Read 165 bytes

All types of MDS commands can be chained. In addition the chain can be concluded with the END command. This command may not be located within a chain.

- INIT, WRITE and READ command, chained

AB	Com-mand	Status	Date		Length	
06	43	00	00	00	80 00	1st command Initialize MDS

AB	Com-mand	Status	Address	Length	Data	
05	41	00	00 F0	05	31 37 33 39 30	2nd command Write 5 bytes

AB	Com-mand	Status	Address	Length	
05	02	00	02 01	1F	3rd command (last command) Read 31 bytes

B.6 Status Byte

The structure of the status byte in acknowledgments and messages from the SLG is described below. Possible error codes which may occur in the acknowledgments and messages are also listed.

Status byte

Bit	7	6	5	4	3	2	1	0

Status	Bit pattern	Bit 7 to 6	=	0
		Bit 5	=	0 (ECC is always on)
		Bit 4 to 0	=	Status code (00 hex to 1F hex)

00	No error Function executed correctly. Message without errors.
01	Presence error: MDS out of field while command still active
02	A queued MDS command was terminated by the “turn off antenna” command.
03	–
04	Error in MDS memory
05	Unknown command/ wrong parameter/ function not permitted
06	Field interference on SLG > The MDS left the field during communication. > Communication between SLG and MDS was terminated by external interference.
07	–
08	–
09	–
0A	–
0B	Memory of MDS cannot be read correctly.
0C	Memory of MDS cannot be written.
0D	Error in specified address (address error) > The specified address does not exist on the MDS. > The command must be checked and corrected in telegram setup. > The MDS is not the right type.
0E	–
0F	–
10	NEXT command not permitted

11	–
12	–
13	SLG doesn’t have enough buffer to store the command.
14	Watchdog message from SLG U92
15	Wrong parameter in RESET function
16	–
17	–
18	Only RESET command permitted
19	Previous command still active
1A	–
1B	Sending job on SLG repeated too often / Data loss possible / RESET command required
1C	Antenna is already off. / Antenna is already on. / Mode in SET-ANT command is unknown. / Antenna is off and the MDS command cannot be executed.
1D	Illegal number of MDSs in the field > Greater than 1 For normal addressing: MOBY I call-compatible (versions 1 and 2) > Greater than bunch specified in RESET command
1E	Wrong number of characters in telegram
1F	Running command terminated by RESET command

C 3964R Procedure

The 3964R procedure offers secure data transmission with a point-to-point connection. Secure data transmission is ensured with block transmission using parity, block check character (BCC) and receipt acknowledgment. A data block may contain all characters from 00 hex to FF hex.

Character frame

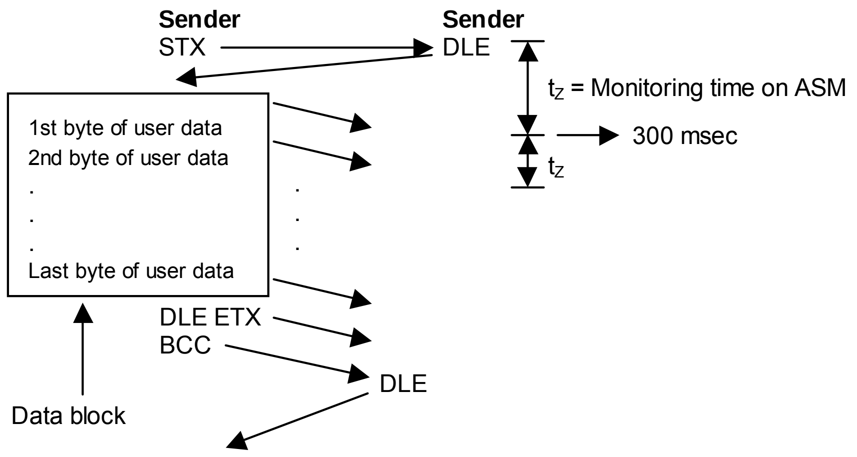
Transmission:	Asynchronous
Baud rate:	9600, 19200, 38400, 57600, 115200 Baud
Data bits:	8
Parity:	Odd
Stop bit:	1

Control character in the 3964R procedure

Table C-1 Control characters in the 3964R procedure

Character	Code (Hex)	Meaning
STX	02	Initialization of a send job (request to send)
DLE ETX	10 03	End of a transmission block
DLE	10	Ready to receive (or DLE doubling in data flow)
NAK	15	Negative response message for block check error or unrecognized start character
DLE DLE	10 10	DLE doubling in data block. Is used when the value 10 hex appears in the data flow.

Block transmission procedure



D Terms/Abbreviations, List of Literature

D.1 Terms/Abbreviations

ASM	Interface module
CHN	Channel number
DI	Digital inputs
DO	Digital outputs
ECC	Error Correction Code
ID	Identification
IP	Internet Protocol
MDS	Mobile data memory
NAK	Negative Acknowledge
PC	Personal Computer
SIM	Serial Interface Module
SLA	Read/write antenna
SLG	Read/write device
SW	Software
TCP/IP	Transmission Control Protocol/Internet Protocol

D.2 List of Literature

/01/ ASM 420 Interface Module Technical Description
6GT2 097-3AF00-0DA2

/02/ SIM Serial Interface Module Technical Description
6GT2 097-3AD00-0DA2

/03/ MOBY F Manual on Configuration, Installation and Service
6GT2 497-4BA00-0EA2

/04/ MOBY E Manual on Configuration, Installation and Service
6GT2 397-4BA00-0EA2

/05/ MOBY I Manual on Configuration, Installation and Service
6GT2 097-4BA00-0EA2

/06/ MOBY U Manual on Configuration, Installation and Service
6GT2 597-4BA00-0EA2

/07/ MOBY user's guide "3964R protocol under Windows NT 4.0/95"
(Included on the MOBY Software floppy disk)