EDIABAS

Electronic Diagnostic Basic System

TRANSPARENT MODE
INTERFACE DESCRIPTION

VERSION 6a

CONTENTS

1. Update History

Version 3.0 First version

Version 3.0A

New: SETZE_TRAP_MASK_REGISTER
New: LIES_TRAP_MASK_REGISTER

Version 3.0B

Correction: SETZE_SG_PARAM_ZURUECK instead of SETZE_SG_PARAMETER_ZURUECK

Version 4.1 revised for V4.1.0

Version 5 revised for EDIABAS V5.1.0

Version 6 revised for EDIABAS V6.0.0

Version 6a revised for EDIABAS V6.4.4

2. Introduction

2.1. About this Manual

This manual describes the interface of the EDIABAS transparent mode. It has been written for use by developers of diagnostic software who work with the API interface, and is based on the API Interface Description [1] and the EIDBSS documentation [2]. API functions and interface functions are only detailed so far as is necessary for an understanding of the transparent mode. The emphasis is on descriptions of the individual TMODE functions. These functions are the jobs provided by the special description file for the transparent mode with the name "TMODE". You will find general information about EDIABAS and control unit description files in Reference [4].

2.2. Conventions

The following typographical conventions are used in this manual:

Example	Description
SAMPLE.C	Upper case characters are used for filenames, registers and operating system commands.
job, string, while	Bold type is used for key words and operators of the BEST/2 and BEST/1 languages and for API functions. In syntax descriptions these words must be written as shown.
expression	Italics designate placeholders for values to be entered by the programmer; e.g., file names.
[option]	Words enclosed in square brackets may be optionally specified.
{ **result** | **argument** }	Curvy braces and vertical strokes characterize entries from which only one must be selected, except when in square brackets.
[constant...] job...	An ellipsis (three dots) which directly follows an expression indicates that several expressions of the same type can follow.
`hallo="Test";`	This syntax designates examples, user entries, program outputs and error messages.

`while() {`
`.`
`.}`
	A column or a row comprising three dots indicates that a section of an example was intentionally omitted.

[1]	Reference to a document in References.

2.3. Special features, definitions, acronyms

The abbreviations used in this and all other EDIABAS documents are explained in the "GLOSSARY" section of the "EDIABAS User Manual".

3. General

The transparent mode of EDIABAS enables the developer of diagnostic/coding software to directly access the interface and control units with the help of a special description file. With the transparent mode, the data that are interchanged between the interface and the application programme or between the control unit and the application programme are not processed in a control unit-specific description file but in the application programme itself.

This means that the diagnostic data are accessed at message level and not at symbolic level. EDIABAS still performs error handling however (e.g. communication errors). With the aid of the transparent mode it is therefore possible to export existing diagnostic software to EDIABAS without having to make structural changes to the existing application programme.

The description file known as "TMODE" provides basic utilities for the transparent mode. These utilities are referred to in the sections that follow as TMODE functions.

TMODE functions for accessing control units:

- Reset control unit parameters in the interface
- Set control unit parameters in the interface
- Set message answer length
- Get keybytes from control unit
- Send and receive a message
- Start repeated send and receive a message
- Get the current answer message after starting the repeated send of a message
- Stop repeated send and receive a message

TMODE functions for accessing the diagnostic bus interface:

- Reset interface
- Read out the interface type
- Read the interface version number
- Read out the voltage at terminal 30
- Read out the voltage at terminal 15
- Read in analog and digital values
- Set digital outputs
- Set the programming voltage
- Trigger the SI relay
- Test the diagnostic lead
- Read out the interface status
- Send a random byte string to the interface

4. The Programming Interface

The programming interface used for accessing TMODE functions consists of following API functions:

- **ApiJobData/apiJobExt**
- **apiResultBinary**

4.1. The API functions apiJobData/apiJobExt

Access to the interface and beyond it to a control unit is possible by sending an API job to EDIABAS. In the transparent mode this job always has the following structure:

apiJobData(*description file, job, data buffer, data length, result* **)**

or

apiJobExt(*description file, job, standard data buffer, standard data length, data buffer, data length, result, reserved)*

The *description file* of the transparent mode is called "TMODE", and makes the link to the TMODE functions. Each TMODE function is implemented in the description file as a separate job.

The name of the TMODE function (e.g. "SENDE_TELEGRAMM") is entered as the *job* (see the section on "TMODE Functions").

Data required by the TMODE function are entered in a *data buffer* (e.g. message data 35,00,05,00 for reading the identification data from the LSM control unit). *Data length* gives the number of data bytes (e.g. number of data is 4 for reading the identification data from the LSM control unit).

In the transparent mode "" is always entered as the parameter *result* because this parameter is not evaluated in the TMODE description file.

Using the API function *apiJobExt standard data buffer* must be set on “”, *standard data length* must be set on 0 and *reserved* must be set on 0.

In transparent mode the same description file ("TMODE") is used for all control units.

EDIABAS - TRANSPARENT MODE

Syntax:

void apiJobData (**char** **ecu*, **char***job*,
unsigned char*parabuf*,
int*paralen*,
char*result***)**

Parameters:

ecu	Name of transparent mode description file: "TMODE"
job	Name of the TMODE function
parabuf	Data bytes. The data bytes depend on the TMODE function.
paralen	Number of data bytes (maximum number: APIMAXPARA)
result	Results to be identified (max. length of result string: **APIMAXRESULT**). Several results must be separated by a semi-colon. A blank string ("") must be defined to process all results. The results depend on the TMODE function.

Return: -

Example:

```
unsigned char  message[APIMAXPARA]={0x35,0x00,0x05,0x00};
int            messagelength=4;

apiJobData("TMODE","SENDE_TELEGRAMM",
           message,message_length,"");
```

EDIABAS - TRANSPARENT MODE

Syntax:

void apiJobExt(**char** **ecu,* **char** **job,*
unsigned char **stdparabuf,* **int** *stdparalen,*
unsigned char **parabuf*, **int** *paralen*,
char **result,* **long** *reserved*)

Parameters:

ecu	Name of transparent mode description file: "TMODE"
job	Name of the TMODE function
stdparabuf	Data bytes for standard jobs. This parameter is not supported by TMODE: ""
stdparalen	Number of data bytes for standard jobs. This parameter is not supported by TMODE: 0
parabuf	Data bytes. The data bytes depend on the TMODE function.
paralen	Number of data bytes (maximum number: APIMAXPARA)
result	Results to be identified (max. length of result string: **APIMAXRESULT**). Several results must be separated by a semi-colon. A blank string ("") must be defined to process all results. The results depend on the TMODE function.
reserved	reserved: always 0

Return: -

Example:

```
unsigned char  telegram[APIMAXPARA]={0x35,0x00,0x05,0x00};
int  telegramLength=4;


apiJobExt("TMODE","SENDE_TELEGRAMM","",0,
           telegram,telegramLenght,"",0L);
```

4.2. The API function apiResultBinary

In the transparent mode, the application programme gets the results of a job sent by the function apiJobData/apiJobExt with the function

apiResultBinary(*target address buffer, target address length, result, result set* **)**

The *target address buffer* is the address of a variable in the application programme (field for data bytes) where EDIABAS stores the result. The *target address length* is the address of an APIWORD variable in the application programme where EDIABAS stores the number of bytes received. The name of the result to be read must be entered as the parameter *result* (not case sensitive). The result names are defined in the separate TMODE functions. All results are in *result set 1*.

The error status is affected, i.e. if an error occurs during processing then the error number and error text can be polled by API functions (see [1]).

APIBOOL apiResultBinary(APIBINARY**buf*,**APIWORD****buflen*,**char****result*,
APIWORD *set***)**

Parameters:

buf	Address of buffer target variable
buflen	Address of length target variable. Up to APIMAXBINARY characters can be transmitted
result	Name of the result
set	Result set number (always 1)

Return:

APITRUE	Result exists
APIFALSE	Job failed or result not present

Example:

```
APIBINARY  message[APIMAXPARA];
APIWORD  message length;
apiResultBinary(message, &message length, "SG_ANTWORT",1);
```

4.3. Application example

In this section an example in language C is used to demonstrate the part of the application programme in which utilities are requested by EDIABAS.

The first part sets the control unit parameters. No result is polled after these parameters are set because this utility does not return a result.

The second part requests the control unit's identification data. After the message is sent the control unit's answer message is polled.

First, a job is always sent using the API function **apiJobData/apiJobExt**. Then **apiState** is called cyclically in a loop until EDIABAS has finished processing the job. Actions such as keyboard polling can be executed in this loop again and again. When the job is finished the result is polled with **apiResultBinary**. Depending on the return value of **apiResultBinary** (TRUE or FALSE), the system continues processing the result or error handling is carried out. If no result is expected, **apiState** checks whether an error has occurred when processing is finished.

```
unsigned char      cu parameter[]= {
                   0x01,0x01,0x01,0x01,0x0F,0x20,0x03,0x64,0x00};
int                cu parameterlength= 9;
unsigned char      request mes[]= {0x35,0x00,0x05,0x00};
int                request mes length = 4;
APIBINARY          answer mes[APIMAXBINARY];
APIWORD            answer mes length;

apiJobData("TMODE",
           "SETZE_SG_PARAMETER_EIDBSS",
           cu parameter, cu parameter length,
           "");

while (apiState() == APIBUSY) {
 /* short programme part, e.g. keyboard polling */
}

if (apiState() == APIREADY) {
 /* continue processing result or continue programme */
}
else {
 /* error handling, e.g. with apiErrorCode */
}

apiJobData("TMODE","SENDE_TELEGRAMM",
           request mes,request mes length,
           "");

while (apiState() == APIBUSY) {
 /* short programme part, e.g. keyboard polling */
}

if (apiResultBinary(answer mes, &answer mes length, "SG_ANSWER", 1))
{
 /* continue processing answer message */
}
else {
 /* error handling, e.g. with apiErrorCode */
}
```

5. The TMODE Functions

An essential part of the transparent mode is a description file in which all TMODE functions (e.g. set control unit parameters, send a message etc.) are imaged on jobs. In this section each job is described by its name, the required parameters, the result name and the result content.

Assignment of API function parameters to names in the job description:

```
apiJobData    ("TMODE",
              Job                        <------------>  Jobname
              Parameter buffer           <------------>  Parameter
              Parameter buffer length    <------------>  Parameter
              Result                     <------------>.."" or result name
        )

apiResultBinary(
              Target address buffer      <------------>  Result content
              Target address length,
              Result                     <------------>  Result name
              Result set                 <------------>  1
        )
```

A string of characters of the *unsigned char* type is always specified as parameters. To specify variables of the int, long etc. type, they must first be converted, in which case the first character is the least significant and the last is the most significant character (Intel format).

The results are also specified as a string of characters of the *APIBINARY* type. If the results are to be interpreted as variables of the int, long etc. type, they must also be converted first. The results are also specified in Intel format.

5.1. Overview of all TMODE functions

- INITIALISIERUNG
- SETZE_INTERFACE_ZURUECK
- SETZE_SG_PARAM_ZURUECK
- SETZE_SG_PARAMETER_ALLG
- SETZE_SG_PARAMETER_EIDBSS
- SETZE_ANTWORTLAENGE
- HOLE_KEYBYTES
- SENDE_TELEGRAMM
- SENDE_TELEGRAMM_WIEDERHOLT
- HOLE_ANTWORT_TELEGR
- STOPPE_WIEDERH_ANFORDERUNG
- LESE_INTERFACE_TYP
- LESE_INTERFACE_VERSION
- LESE_SPANNUNG_KL30
- LESE_SPANNUNG_KL15
- LESE_PORT
- SETZE_PORT
- SETZE_PROGRAMMIERSPANNUNG
- SETZE_SIA_RELAIS
- TESTE_DIAGNOSELEITUNG
- HOLE_INTERFACE_STATUS
- REICHE_AN_INTERFACE_DURCH
- SETZE_TRAP_MASK_REGISTER
- LIES_TRAP_MASK_REGISTER

5.2. INITIALISIERUNG

Jobname:	INITIALISIERUNG
Parameters:	None
Result name:	DONE
Result contents:	Value 1 as a 2-byte number
Description:	This job is called automatically whenever the description file is loaded or an EDIABAS error has occurred. It defines which

interface is connected. This job requires the result DONE that is interpreted by the EDIABAS system. In the transparent mode this result is always 1.

5.3. SETZE_INTERFACE_ZURUECK

Jobname: SETZE_INTERFACE_ZURUECK

Parameters: None

Result name: -

Result contents: None

Description: This job puts the interface in the initialising state and tests the diagnostic interface. The EDIC (IDBSS) will not accept a command for about 2 seconds after the job.

5.4. SETZE_SG_PARAM_ZURUECK

Jobname: SETZE_SG_PARAM_ZURUECK

Parameters: None

Result name: -

Result contents: None

Description: This job breaks off communication with a control unit and cancels the communication parameters. Any message from the control unit still stored in the EDIC (IDBSS) is lost.

5.5. SETZE_SG_PARAMETER_ALLG

Jobname: SETZE_SG_PARAMETER_ALLG

Parameters: The communication parameters are defined as the parameters.

You will find a detailed description of the parameters in [7] in the description of the **set_communication_pars** function.

Result name: -

Result contents: None

Description: This job sets the communication parameters required for communicating with a control unit. The parameter format is independent from the interface. Once parameters are set with this job it is not necessary to change the user software when there is a change of interface. The job must be called before any new control unit is addressed. As well as the communication parameters the message leader is filled with default values in EDIABAS depending on the set concept. See 5.7 for further details about message leaders.

5.6. SETZE_SG_PARAMETER_EIDBSS

Jobname: SETZE_SG_PARAMETER_EIDBSS

Parameters: The communication parameters are defined as the parameters. The parameters must have the format requested by the EIDBSS application on the EDIC, see [2]. The check bytes are not transferred.

Result name: -

Result contents: None

Description: This job sets the communication parameters required for communicating with a control unit. The parameter format is dependent on the interface. Once parameters are set with this job the user software must be changed when there is a change of interface. The job must be called before any new control unit is addressed. As well as the communication parameters the message leader is filled with default values in EDIABAS depending on the set concept. See 5.7 for further details about message leaders.

5.7. SETZE_ANTWORTLAENGE

Jobname: SETZE_ANTWORTLAENGE

Parameters: The message leader is defined as the parameter. The message leader consists of two parameters, each comprising two bytes. The first parameter is the answer length and the second is the answer offset.

Answer length: This indicates the length of the anticipated CU answer.

concepts 1, DS1, DS2 and concept 3:

positive: number of anticipates bytes in answer message from CU

(constant answer length)

negative: position of answer length in answer message (from byte 0)

(variable answer length)

concept 2 and 4:

Maximum number of blocks that contain the desired information. If zero is specified as the answer length then all answer blocks including the first acknowledge block will be collected up.

Answer offset: Variable answer length with concept 1, DS1, DS2 only. The answer length is computed as follows:

Length = (answer length + 1) + answer offset

Default value for the message leader after the control unit parameters are set:

Concept	Length	Offset
1	-2	0
3	52	0
5,6(DS1,DS2)-1	0	
2,4	1	0
others	1	0

Result name: -

Result contents: None

Description: When it sends a message the interface needs a message leader containing information about the anticipated answer length and answer offset. This information is the same for most CU messages. By setting the leader with SETZE_ANTWORTLAENGE the user can avoid having to transmit it with every message. The leader is automatically set at the beginning of the message when it is sent. If the user does not set a leader the system uses the default values used by EDIABAS to set the communication parameters.

5.8. HOLE_KEYBYTES

Jobname: HOLE_KEYBYTES

Parameters: None

Result name: KEYBYTES

Result contents: Control unit keybytes

Description: This job reads the key bytes and identification data from a concept 2, concept 3 or a concept 4 control unit. The control unit is woken up automatically if it has not yet been triggered.

5.9. SENDE_TELEGRAMM

Jobname: SENDE_TELEGRAMM

Parameters: Control unit message according to the CU specifications. The checksum is omitted with concept 2, DS1 and DS2 control units. The ETX at the block end is omitted with concept 2 and concept 4 control units.

Result name: SG_ANTWORT

Result contents: Control unit answer. With concept 2 and 4 control units the answer blocks are appended to each other, omitting the last byte of each block (ETX).

Description: This job sends a message to a control unit and gets the answer.

5.10. SENDE_TELEGR_WIEDERHOLT

Jobname: SENDE_TELEGR_WIEDERHOLT

Parameters: Control unit message according to the CU specifications. The checksum is omitted with concept 2, DS1 and DS2 control units. The ETX at the block end is omitted with concept 2 and concept 4 control units.

Result name: -

Result contents: None

Description: This job sends the defined control unit message to the control unit repeatedly. This mode can be terminated with the job STOPPE_WIEDERH_ANFORDERUNG. In this mode a renewed call of SENDE_TELEGR_WIEDERHOLT or a call of SENDE_TELEGRAMM is answered with the error message IFH_0006.

5.11. HOLE_ANTWORT_TELEGR

Jobname: HOLE_ANTWORT_TELEGR

Parameters: None

Result name: SG_ANTWORT

Result contents: Control unit answer message. With concept 2 and 4 control units the answer blocks are appended to each other, omitting the last byte of each block (ETX).

Description: If the repeated request for control unit answers has been started with SENDE_TELEGR_WIEDERHOLT, the answers can now be polled with this job. It is always the current CU answer which is polled.

5.12. STOPPE_WIEDERH_ANFORDERUNG

Jobname: STOPPE_WIEDERH_ANFORDERUNG

Parameters: None

Result name: -

Result contents: None

Description: This job stops the repeated sending and receiving of messages.

5.13. LESE_INTERFACE_TYP

Jobname: LESE_INTERFACE_TYP

Parameters: None

Result name: TYP

Result contents: "IDBSS" or "EIDBSS"

Description: This job polls the interface type as a zero-terminating string. "EIDBSS" is the name of the application (diagnostic software) on the EDIC.

5.14. LESE_INTERFACE_VERSION

Jobname: LESE_INTERFACE_VERSION

Parameters: None

Result name: VERSION

Result contents: Version number as low and high byte

Description: This job polls the version number of the interface.

5.15. LESE_SPANNUNG_KL30

Jobname: LESE_SPANNUNG_KL30

Parameters: None

Result name: SPANNUNG

Result contents: Four byte value that indicates voltage in mV.

Description: This job polls the voltage at terminal 30 in mV.

5.16. LESE_SPANNUNG_KL15

Jobname: LESE_SPANNUNG_KL15

Parameters: None

Result name: SPANNUNG

Result contents: Four byte value that indicates voltage in mV.

Description: This job polls the voltage at terminal 15 in mV.

5.17. LESE_PORT

Jobname: LESE_PORT

Parameters: One byte indicating the port number.

Result name: PORTWERT

Result contents: Four byte value indicating the port value.
Port 0-5: analog input 0 - 5 (voltage in mV)
Port 6: analog input T 15 (voltage in mV)
Port 7: analog input T 30 (voltage in mV)
Port 8: jumper field (digital value)

Description: With EDIC nine ports can be read. Ports 0 to 7 are analog inputs, port 8 polls the value of the jumper field.

5.18. SETZE_PORT

Jobname: SETZE_PORT

Parameters: The first byte indicates the port number. Bytes 2 to 4 indicate the value at which the port is set.
Port 9: digital outputs

Result name: -

Result contents: None

Description: Ports can be set with this job. Only port 9 (digital outputs) can be set in EDIC.

5.19. SETZE_PROGRAMMIERSPANNUNG

Jobname: SETZE_PROGRAMMIERSPANNUNG

Parameters: Four byte value giving the programming voltage in mV.

Result name: -

Result contents: None

Description: This job turns the programming voltage on and off. The programming voltage level can be given in mV (0 - 3300 mV). A 0 V voltage means programming voltage is "off", otherwise "on".

5.20. SETZE_SIA_RELAIS

Jobname: SETZE_SIA_RELAIS

Parameters: Switching time in milliseconds as a 2 byte value.
Switching time = 0x0000: de-energize permanently
Switching time = 0xFFFF: energize permanently

Result name: -

Result contents: None

Description: This job energizes the EDIC service interval relay for the specified time.

5.21. TESTE_DIAGNOSELEITUNG

Jobname: TESTE_DIAGNOSELEITUNG

Parameters: None

Result name: ERGEBNIS

Result contents: Test result as a 2 byte value.
0: Error occurred
1: No error occurred

Description: This job tests the diagnostic lead; there must be a short between the RD and TD leads.

5.22. HOLE_INTERFACE_STATUS

Jobname: HOLE_INTERFACE_STATUS

Parameters: None

Result name: IF_STATUS

Result contents: Status bytes of the interface

Description: This job requests the status bytes of the interface (see [2]).

5.23. REICHE_AN_INTERFACE_DURCH

Jobname: REICHE_AN_INTERFACE_DURCH

Parameters: Job to the interface (see [2])

Result name: IF_ANTWORT

Result contents: Answer from interface

Description: This job passes the data bytes sent to the EDIABAS direct to the interface. EDIABAS does not evaluate the answer from the interface. This job does not evaluate the status byte from the interface.

5.24. SETZE_TRAP_MASK_REGISTER

Jobname: SETZE_TRAP_MASK_REGISTER

Parameters: Sets the trap mask register with the long value defined as parameter 1.

Result name: -

Result contents: None

Description: This job sets the trap mask register that can handle the occurrence of an error in the description file.

5.25. LIES_TRAP_MASK_REGISTER

Jobname: LIES_TRAP_MASK_REGISTER

Parameters: -

Result name: TMR

Result contents: Current value of the trap mask register

Description: This job reads the trap mask register that can handle the occurrence of an error in the description file.

A. LIST OF REFERENCES

[1] EDIABAS: API Interface Description

[2] SOFTWARE DOCUMENTATION FOR EIDBSS-BMW CU COMMUNICATION INTERFACE
Softing GmbH, Version 1.4

[3] EDIABAS: BEST/1 - Language and Interpreter

[4] EDIABAS: User Manual

[5] EDIABAS: BEST/2 - Language Description

[6] EDIABAS: API User Manual

[7] EDIABAS: BEST/2 - Function Primer