EDIABAS

Electronic Diagnostic Basic System

USER MANUAL

INSTALLATION GUIDE

VERSION 6c

Contents

Contents **2**

1. Revision history **5**

2. Introduction **6**

2.1. About this manual 6
2.2. Notational conventions 6
2.3. Special features, terms, acronyms 7
2.4. Trademarks 7

3. General **8**

3.1. Diagnostics and coding of ECUs 8
3.2. Problem definition 8
3.3. Approaches to problem solving 10
3.4. Implementing solutions with EDIABAS 12
3.5. Structure 12
3.6. Job concept 15
3.7. Description language BEST 15
3.8. Sequences 16
3.9. Availability 19

4. Installation and administration **20**

4.1. Delivery packages 20
4.1.1. Delivery package RUNTIME SYSTEM 20
4.1.1.1. Files for WIN32 20
4.1.1.2. Files for WIN16 22
4.1.1.3. Files for WINCE 22
4.1.1.4. Files for SCO-UNIX 23
4.1.1.5. Files for QNX 23
4.1.1.6. Documentation 24
4.1.2. Delivery package APPLICATION DEVELOPMENT KIT 24
4.1.2.1. Files for WIN32 25

4.1.2.2. Files for WIN16 25
4.1.2.3. Dateien unter WINCE 25
4.1.2.4. Files for SCO-UNIX 27
4.1.2.5. Files for QNX 28
4.1.2.6. Documentation 29
4.1.3. Delivery package BEST DEVELOPMENT KIT 29
4.1.3.1. Files for WIN32 30
4.1.3.2. Files for WIN16 31
4.1.3.3. Documentation 31
4.2. System requirements 32
4.2.1. MS-WINDOWS 32
4.2.2. MS-WINDOWS CE 32
4.2.3. SCO-UNIX 32
4.2.4. QNX 32
4.3. Installing for the first time 33
4.3.1. MS-WINDOWS 33
4.3.2. SCO-UNIX 33
4.3.3. QNX 34
4.4. EDIABAS program paths 35
4.4.1. MS-WINDOWS 35
4.4.2. SCO-UNIX 35
4.4.3. QNX 35
4.5. EDIABAS configuration 36
4.5.1. Search sequence of EDIABAS.INI 36
4.5.2. Configuration overview 37
4.5.3. Select the hardware interface 39
4.5.4. MS-WINDOWS 39
4.5.5. SCO-UNIX 40
4.5.6. QNX 40
4.5.7. Directory of ECU description files 40
4.5.8. Character Set Conversion 40

4.5.9. WIN16/32-Umsetzung 41

4.6. Start and terminate EDIABAS 41

4.6.1. WIN32 42

4.6.2. WIN16 42

4.6.3. WINCE 42

4.6.4. SCO-UNIX 42

4.6.5. QNX 43

4.7. Un-install 43

4.7.1. MS-WINDOWS 43

4.7.2. MS-Windows CE 44

4.7.3. SCO-UNIX 44

4.7.4. QNX 44

4.8. Protection mechanism 45

4.9. Remote diagnostics 46

4.9.1. EDIABAS.INI on the local PC 46

4.9.2. EDIABAS.INI on the remote-controlled PC 48

4.9.3. Procedure for remote diagnostics 48

4.10. EDIABAS Parallel Operation 48

5. Help programs 50

5.1. UINSTALL 50

5.1.1. SCO-UNIX 50

5.1.2. QNX 50

5.2. PE (Password Editor) 51

5.3. DEVCLOSE 52

5.4. BESTINFO 52

5.5. BESTVER 54

5.6. XTRACT 55

5.7. STRIP 57

A. GLOSSARY 58

1. Revision history

Version 3.0	Created overview and User Manual/Installation Guide
Version 3.0A	Combined above documents
Version 3.0B	Added Rev., Glossary and Introduction
Version 4.1	Revised for EDIABAS V4.1.0
Version 5	Revised for EDIABAS V5.1.0
Version 5a	Extended for EDIABAS V5.5.0
Version 5b	Extended for remote diagnostic Win32
Version 5c	Configuration for IFH Trace
Version 5d	Revised the XTRACT output functionality
Version 5e	Extended for QNX
Version 6	Extended for EDIABAS V6.0.0
Version 6c	Revised for EDIABAS V6.4.4

2. Introduction

2.1. About this manual

This manual describes how to operate the **EDIABAS** (**E**lectronic **D**iagnostic **B**asic **S**ystem).

2.2. Notational conventions

The following typographical conventions are used throughout this manual:

Example	Description
SAMPLE.C	Uppercase denotes file names, registers and operating system commands.
apiJob, APIREADY	Bold-faced type identifies keywords and operators of the language BEST/2 and BEST/s as well as the API functions.. These words must be written exactly as specified in syntax descriptions.
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, terms, acronyms

An explanation of abbreviations used in this and all other EDIABAS documentation can be found in chapter "GLOSSARY".

2.4. Trademarks

Microsoft, MS, MS-DOS, WINDOWS, WIN32 and Pocket PC are registered trademarks of the Microsoft Corporation.

SCO, SCO UNIX and OpenServer are registered trademarks of Santa Cruz Operation, Inc.

QNX is a registered trademark of QNX Software Systems Ltd.

3. General

3.1. Diagnostics and coding of ECUs

ECUs (**E**lectronic **C**ontrol **U**nits), are developed by various participating members including component suppliers of the Automobile Industry, vehicle manufactures, themselves, and contractors. The ECUs tested and produced by suppliers are installed in the vehicle at the automobile manufacturer, if applicable coded (programmed) specifically for the vehicle via their diagnostic interface and tested for proper functioning in the vehicle. Whenever an error occurs, the cause of error is determined and remedied by means of "Electronic Diagnostics" accompanied by a computer-based testing system. After the vehicle has been delivered, errors occurring within the scope of "Inspection and Service" are corrected in the shop. Once again, "Electronic Diagnostics" are applied. When an ECU is swapped in the shop, the defective unit is repaired by the manufacturer, and its proper functioning is tested, by means of "Electronic Diagnostics". The newly installed ECU may need to be re-coded again on the premises according to vehicle specifications.

In accordance with this life cycle, coding and testing ECUs is an often recurring and prescribed job duty for:

- Development of the ECU
- Testing of the ECU in the test lab
- Tests in the test vehicle
- Manufacturing of the ECU
- Final inspection of the ECU
- Installing and checking of the ECU in the pre-assembly
- Vehicle-relevant coding of the ECU
- Function test in the vehicle
- Final inspection of the vehicle
- Servicing or debugging in scope of Customer Service
- Replacing and re-programming in the shop
- Repair of the ECU

3.2. Problem definition

If a large number of ECUs which, in part, have been developed and supplied by several manufacturers are employed in several model series of vehicles, an extremely high percentage of ECU fluctuations results, since the on-going advancements of the subsystems occur in the life span of a vehicle series. In comparison, engines can be considered which are employed in several different designs in the same body of a vehicle series.

Due to the discontinuation and new use of ECUs as well as technical changes to ECUs, the modifications required in coding and testing systems (modification, commissioning, test) increase over-proportionally. The expenditure of documentation and the risk of error increase.

Due to intense »dissection« of tasks in large companies and sequential procedures, the steps mentioned above lead to multiple processing, extending from the development of an ECU via production to Customer Service:

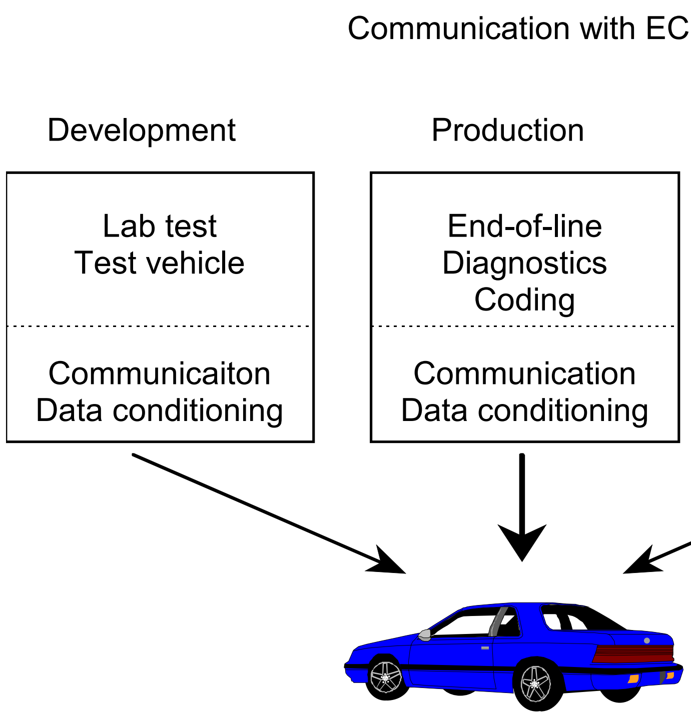

Fig. 3.2-1: Previous state

Coding and testing programs are individually created in diverse areas: At the ECU developer; in the test department; in pre-production; in customer service; ECU repair center. This means, based on the documentation pertaining to an ECU, the steps involving development (or advancement) and testing of programs are repeated more than once! The apparent differences in the job definitions and objectives of the individual phases discourage the view for mutuality, namely each the same component ECU.

3.3. Approaches to problem solving

The identical parts of the system mentioned above can be used multiply by all application programs following a non-recurring development phase. This occurs in sense of a type »**operating system**« developed for ECU communication, for which a matching »**communication driver**« is provided for each ECU.

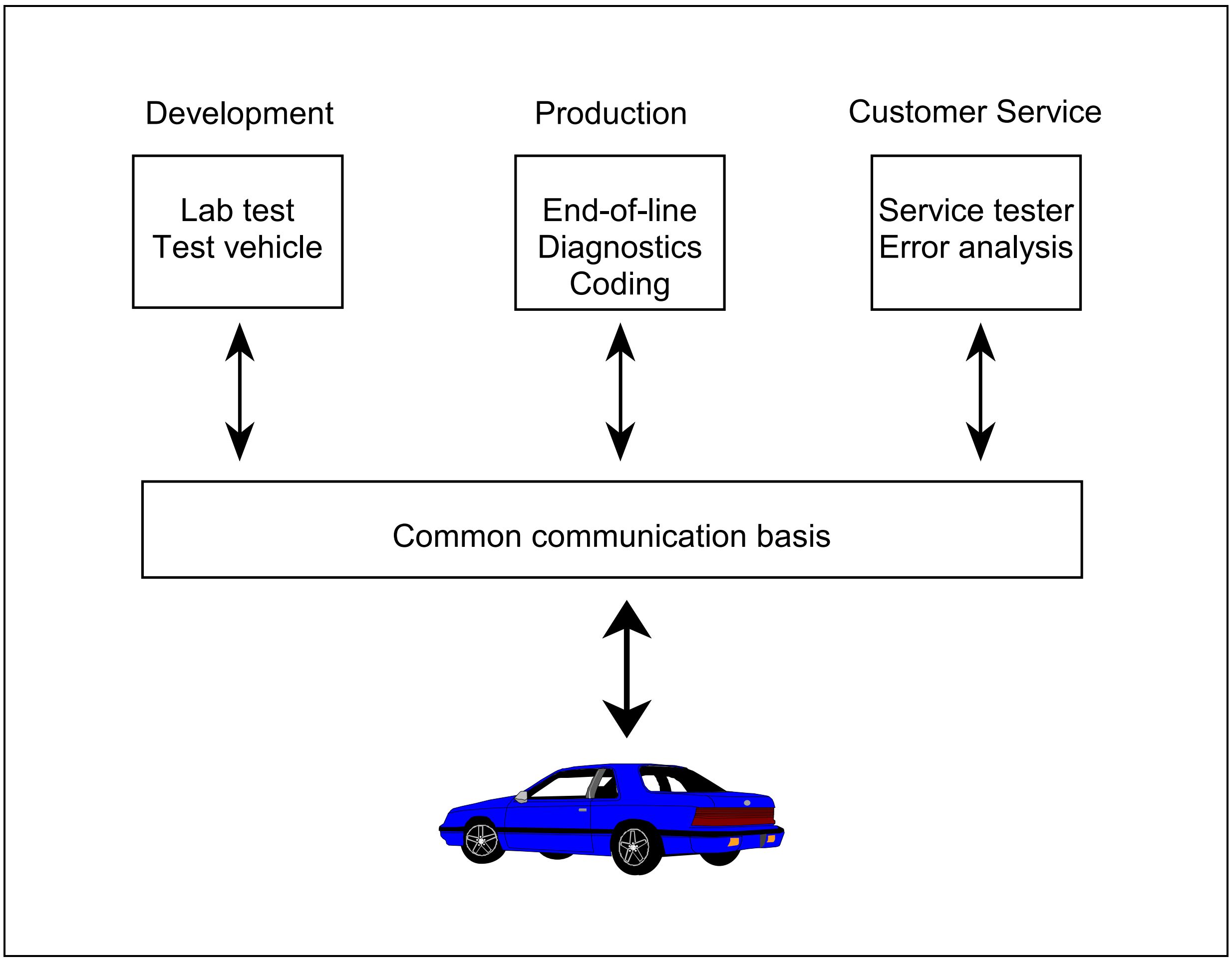

Fig. 3.3-1: Desired state

Hence, similar to how a conventional operating system conceals information behind special functions (e.g., about precise access to hard disks/to the screen), this »operating system« also conceals the exact information about access to an ECU.

This »**operating system**« is not fixed to the application programs or even combined with one another in data regions. Instead, it only provides »**services**«. These services can be issued from the application program and are subsequently processed (when desired) time independently of these. The result of this »service« can then be processed by the application after it has been executed. The **application program (the Client)** and the **operating system (the Server)** only exchange messages and data with one another.

The Server translates, the raw data received from the ECU into the symbolic data requested by the application program.

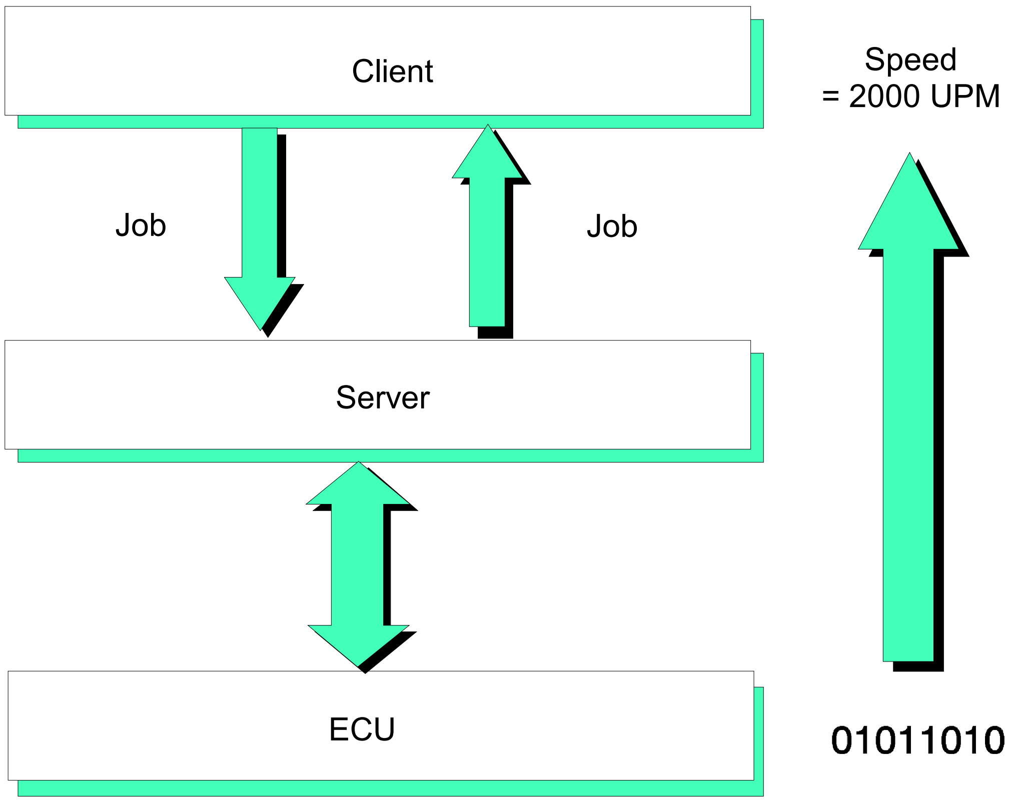

Fig. 3.3-2: Compiler function

In this process, the Server, itself, is not equipped with its own intelligence, but stores its knowledge about the individual ECUs in easily (i.e., even at runtime) exchangeable data modules. These data modules contain the complete (or only the required) knowledge about translation of the binary ECU data into symbolic data.

Each Client (i.e. application program) is provided with a software interface that conceals the operating system-dependent part for exchanging the messages under a uniform procedural interface. Several application programs can send their jobs to the server in time-sharing mode; i.e., as soon as an application program frees the server, the next pending job can be processed.

3.4. Implementing solutions with EDIABAS

All approaches for problem-solving mentioned above have been implemented in EDIABAS. **EDIABAS can therefore be regarded as an »operating system for communication with ECUs in Client/Server architecture**«. Since, however, the term »operating system« would be extremely confusing in this relationship and, in this case, a common basis for other application programs would be created, the term »**Basic System**« has been used by Softing. This "Basic System" in "Electronic Diagnostics" is called:

EDIABAS = **E**lectronic **DIA**gnostic <u>**BAS**ic System</u>

3.5. Structure

EDIABAS has a fixed structure independent of the operating system employed.

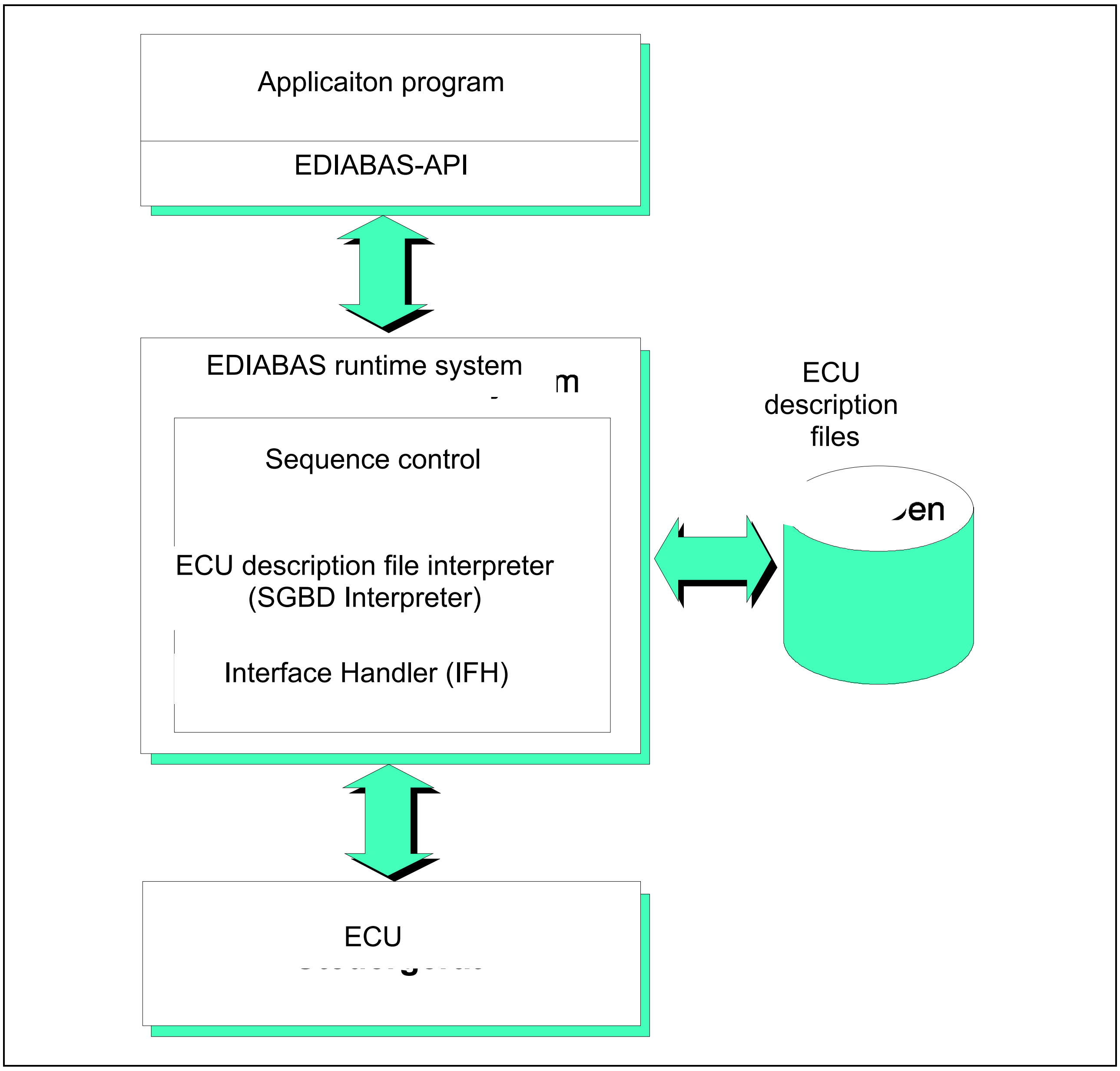

Figure 3.5-1: EDIABAS structure

Figure 3.5-1 illustrates the basic structure of the individual sub-components as well as the data and communication paths.

The ECU-specific knowledge of EDIABAS is stored in the **ECU description files (SGBDs)**. The ECU description files are created "variant orientated"; i.e. exactly one variant-specific ECU description file (SGBD) exists for each ECU variant. A superior, **group ECU description file** can exist for all ECUs of a group (in general, these are

ECUs with the same address). Which variant of this group is connected to the diagnostics bus can be determined using the methods contained in this file.

The ECU description files are loaded and interpreted by the EDIABAS runtime system when instructed by an application program. The file name of the ECU description file (no extension) is the name with which the application programs reference an ECU "variant" or ECU "group".

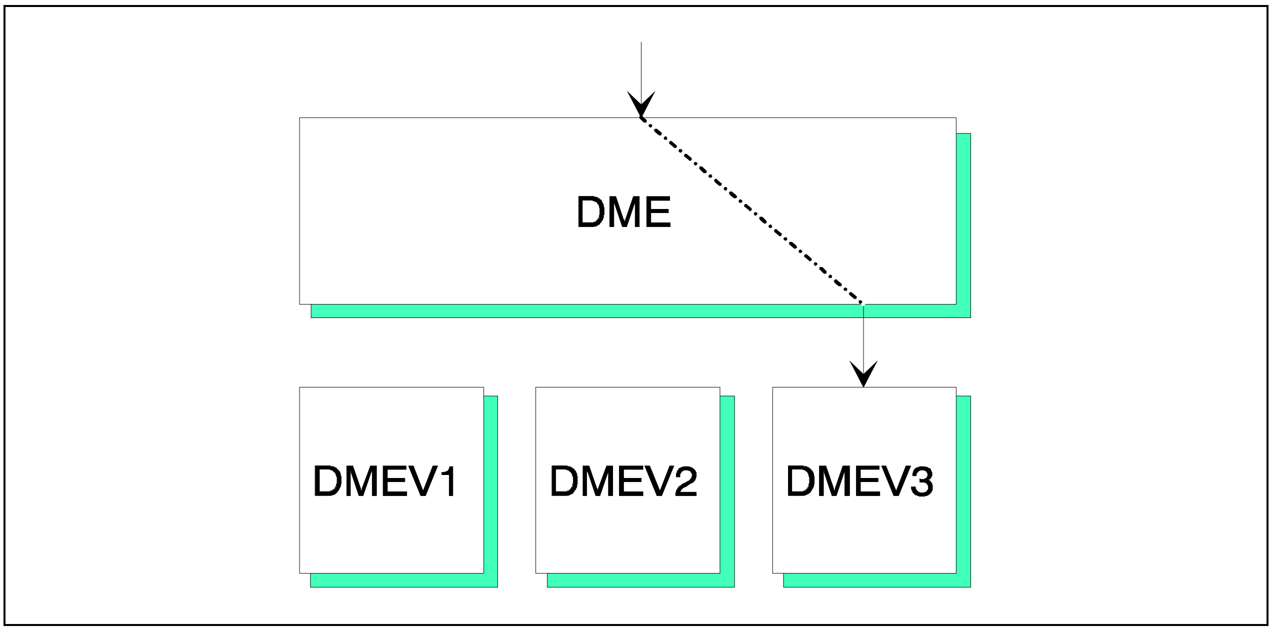

Figure 3.5-2: ECU description file and group ECU description file

The main constituent of EDIABAS is the "**runtime system**". This runtime system can be subdivided into three essential parts:

1. **Sequence control** (kernel)
2. **ECU description file interpreter (SGBD Interpreter)**
3. **Interface Handler (IFH)**

Sequence Control determines the behavior of the entire system and contains the communication interface to the application programs. It is responsible for the file and error management.

The **ECU description file interpreter (SGBD Interpreter)** converts the binary ECU data into symbolic data. It interprets the data, files, methods and sequences contained in the ECU description file, initializes the communication with the ECU and returns the self-determined results of the sequential control for forwarding to application program.

The **Interface Handler (IFH)** is an exchangeable software module which, depending on the hardware interface employed, can be integrated in various versions by the user. A data request to the ECU is transposed by the Interface Handler into a job for the currently applied hardware interface. The data returned from this interface are appropriately filtered from interface-dependent parts and, in this way, can be evaluated independent of the interface hardware used. Even the errors returned from the interface are standardized to the internal error messages. In addition, the Interface Handler provides simulation of ECUs. Thus, EDIABAS-based application programs can also then be tested when the applied ECUs do not exist.

In order to simplify access to EDIABAS, a software interface is linked to each application program. This interface controls access to EDIABAS via a pre-defined procedural interface. This so-called **»Application Programming Interface« (EDIABAS-API)** handles the system-dependent communication of the runtime system. The data received from the runtime system are managed within the EDIABAS-API for the application program. In this way, the runtime system is freed from managing this data, therefore allowing it to be structured considerably compacter.

3.6. Job concept

As already shown in the general approaches of problem-solving, EDIABAS, as Server, offers "services" to the Clients. These services are called **Jobs**. A job is a self-contained task to read and evaluate data from the ECU. It may be called at any time without consideration of the job sequence (principle of independence). After completing processing, the job returns results which can be used by the application program. The jobs are not permanently programmed in EDIABAS, but are a constituent of the "ECU description files" (SGBDs). They are labeled according to their function; e.g. READ_ERRORMEMORY or READ_ENGINESPEED. There are also jobs which are automatically called by the runtime system and therefore "must" or "may" exist in each description file. These include the job INITIALISIERUNG, (initializes the ECU description file), the optional job ENDE (de-initializes the ECU description file) and the job IDENTIFIK,CATION in a group description file (determines the currently installed ECU variant).

3.7. Description language BEST

Jobs are formulated in an own description language for ECUs called **BEST** (**BE**schreibungssprache für **ST**euergeräte). This description language exists in two variations. The first variant is a low-level, assembler-similar language called **BEST/1**. The second variant is the high-language version **BEST/2** with 'C'-like syntax.

In general, all ECU description files are written in BEST/2. In BEST/2, jobs are defined similar to functions; i.e., a job header exists which lists all I/O parameters. Variables and fields can be defined and used within a job. All necessary arithmetic operators are variable. An extensive library contains functions for manipulating data and communicating with the ECU. The language, itself, contains almost all control structures known by 'C'. The ECU description files defined in this way must be converted into a format which can be read by the runtime system. This must be performed using a **compiler (BEST2WIN)** prior to use in EDIABAS. On one hand, this occurs because of performance reasons and, on the other hand, so that syntax errors can be excluded during the runtime.

The **Source Text Debugger BestView** is available to test the compiled BEST/2 description files. By means of BestView, the sequence of a job can be exactly tracked: variable contents can be monitored, displayed and modified.

3.8. Sequences

A job is issued in the application program by means of the API functions **apiJob**, **apiJobData** or **apiJobExt**. These functions require at least two specifications: First, the ECU to be addressed (or the ECU group) and, second, the job which is to be executed.

These two specifications are sent to the EDIABAS runtime system. The runtime system first determines whether this data concerns an ECU "variant" or an ECU "group". If a group was addressed, the job IDENTIFIKATION in the group description file is automatically processed. If a valid ECU variant was determined from the job IDENTIFIKATION, the associated variant file is automatically loaded with the assistance of the result VARIANTE as if the name had already been transferred from the application program. The variant file processes the job requested by the application program. A job may either appear directly in the variant file or in the base file. Base files are ECU description files referenced from the variant file and which can be considered as a part of the variant file. Afterwards, the results of this job are made available to the application program.

Whenever a description file is re-addressed (i.e. after each change), first the job ENDE of the last loaded ECU decryption file is called. This call is made only when the job ENDE exists in the file. It enables the hardware to be de-initialized. Afterwards the job INITIALISIERUNG is called in the new description file. This job must exist in each description file (obligatory) and can be used to initialize the interface hardware. Only now is the specified job executed. When the same job is called again, no standard job is executed anymore.

If an error occurs in one of the sub-components when processing a job, processing is immediately aborted, and an error message is returned to the application program. In this case, any results are deleted. In this situation, the job .INITIALISIERUNG is automatically called again before re-calling the same job.

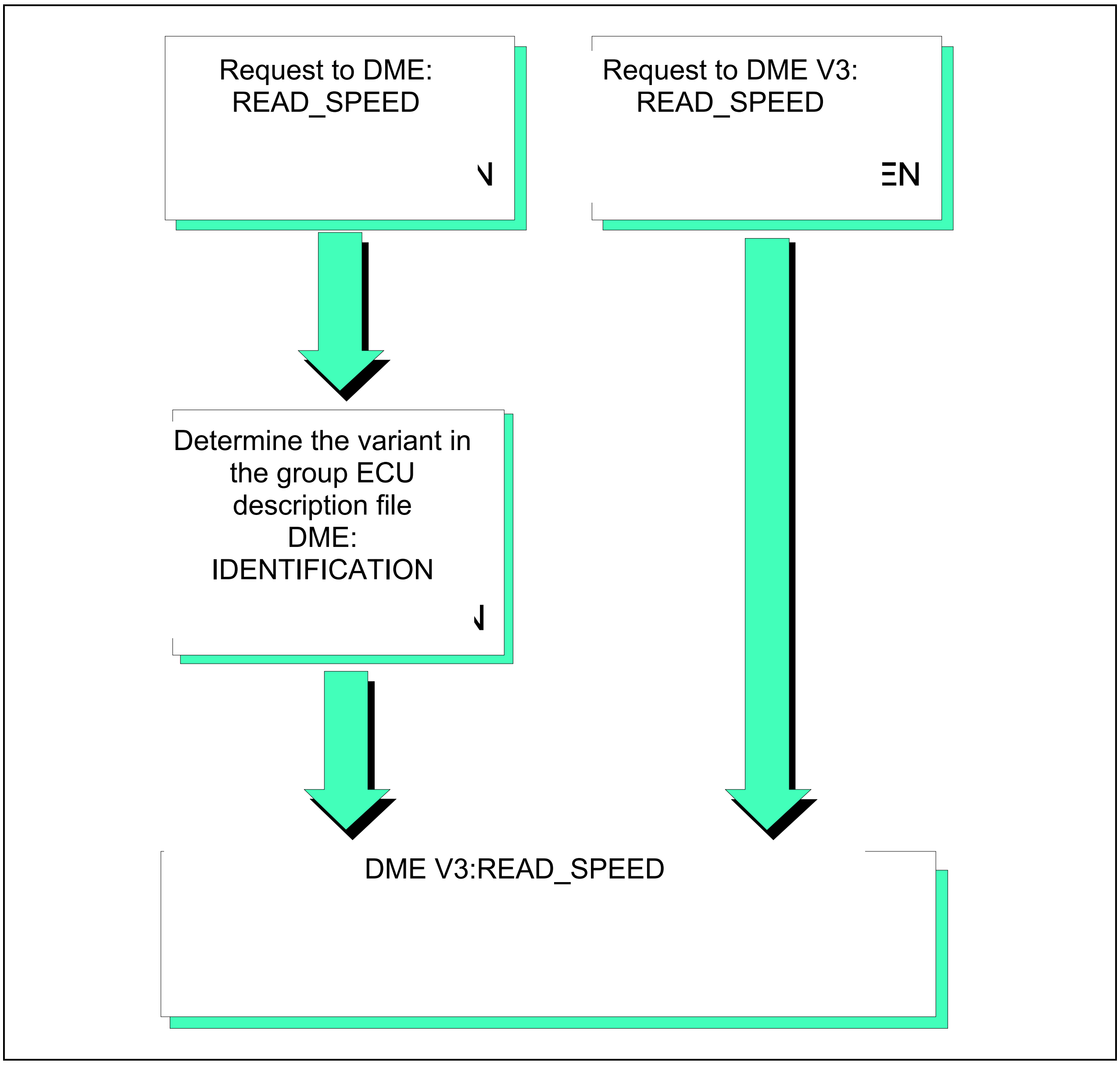

Figure 3.8-1: ECU description file and job selection

As a job is being processed, its commands are sequentially interpreted. Any communication requests to the ECU are forwarded by the Interface Handler and processed. The results determined are sent to the application program.

Application	EDIABAS	ECU

`int data;`
`apiJob("DMEV3",`
`      "READ_SPEED"...);`		

	`job ( name: READ_SPEED;`
	`      result: SPEED;`
	`      type: integer`
	`... ) {`
	`char response[];`
	`send_and_receive(response`
	`,     tel_readspeed);`	

		`-> 59 04 02 03`
		`<- 59 05 03 05 05`

	`SPEED=response[3]*256 +`
	`        response[4];`
	`}`	

`apiResultInt(&variable,`
`.........”SPEED”, set);`
 	
`printf("Speed:`
`%d",variable);`		

Figure 3.8-2: Job sequence

The application program can read the results delivered from EDIABAS by means of the function apiResultXXX, whereby various formats are supported. In this process, data is converted whenever possible. Thus, for example, the result speed (e.g. rpm) which was calculated as integer in the description file can be requested in real format by the application program.

3.9. Availability

The EDIABAS runtime system and the EDIABAS application development environment have been designed as easy-to-port systems. They are presently available for the following platforms:

- MS-WINDOWS 95/98/ME/NT4/2000/XP (WIN32)
- MS-WINDOWS 3.11/95/98/ME (WIN16)
- Pocket PC 2002/2003 (WINCE)

- SCO OpenServer 5
- QNX 4.23

Throughout this manual MS-WINDOWS is used for WIN32 and WIN16.

Throughout this manual MS-WINDOWS CE is used for Pocket PC 2003 and Pocket PC 2002

The development environment for ECU description files is only available under MS-WINDOWS.

Throughout this manual SCO-UNIX is used for SCO OpenServer 5.

I

4. Installation and administration

4.1. Delivery packages

A total of three delivery packages are currently available. The scope of function is explained in detail in the following sections:

1. **RUNTIME SYSTEM**
2. **APPLICATION DEVELOPMENT KIT**
3. **BEST DEVELOPMENT KIT**

4.1.1.Delivery package RUNTIME SYSTEM

The delivery package RUNTIME SYSTEM available for all systems listed in section "Availability". This package is the basis for all other EDIABAS delivery packages. RUNTIME SYSTEM must **always** be installed.

The delivery package allows application programs to run diagnostic sequences via EDIABAS.

4.1.1.1. Files for WIN32

The RUNTIME SYSTEM contains the following files:

README32	Installation notes
BIN\<Paßwortdatei>	Name of password file (8 characters). The file has no extension. Example: 07DE3473
BIN\PE.EXE	Password Editor (PE)
BIN\ANSI2OEM.TAB	Codemapping table
BIN\EDIABAS.INI.Example	EDIABAS configuration file example
BIN\EBAS32.DLL	EDIABAS runtime system
BIN\EBAS32.EXE	EDIABAS visualization
BIN\API32.DLL	Access to the EDIABAS runtime system
BIN\APIVB32.DLL	Access to the EDIABAS runtime system for Visual Basic
BIN\TRACEX32.EXE	Trace server
BIN\BESTINFO.EXE	Dispaly of BEST object file contents
BIN\BESTVER.EXE	Version test of BEST object files
BIN\XTRACT.EXE	Display of the help texts of BEST object files

BIN\STRIP.EXE	Remove the debug and help texts in BEST object files
BIN\IFHSRV32.EXE	IFH server for remote diagnostics
BIN\NMSIFH32.DLL	IFH service for remote diagnostics
BIN\VMC32.DLL	Access to connecting management for remote diagnostics
BIN\XEDIC32.DLL	EDIC-IFH
BIN\XREMOT32.DLL	IFH for remote diagnostics
BIN\NETTCP32.DLL	Communication DLL for remote diagnostics
BIN\NETPRO32.DLL	Communication DLL for remote diagnostics
ECU\TMODE.PRG	ECU description file for transparent mode

4.1.1.2. Files for WIN16

The RUNTIME SYSTEM contains the following files:

README	Installation notes
BIN\PE.EXE	Password Editor (PE)
BIN\ANSI2OEM.TAB	Codemapping table
BIN\EDIABAS.INI	EDIABAS configuration file
BIN\EDIABASW.EXE	EDIABAS runtime system
BIN\API.DLL	Access to the EDIABAS runtime system
BIN\APIVB.DLL	Access to the EDIABAS runtime system for Visual Basic
BIN\BESTINFO.EXE	Dispaly of BEST object file contents
BIN\BESTVER.EXE	Version test of BEST object files
BIN\XTRACT.EXE	Display of the help texts of BEST object files
BIN\STRIP.EXE	Remove the debug and help texts in BEST object files
BIN\XEDIC.DLL	EDIC-IFH
ECU\TMODE.PRG	ECU description file for transparent mode

4.1.1.3. Files for WINCE

The RUNTIME SYSTEM contains the following files:

BIN\<passwordfile>	Name of password file (8 characters). The file has no extension. Example: 07DE3473
BIN\EBASCE.DLL	EDIABAS runtime system
BIN\APICE.DLL	Access to the EDIABAS runtime system
BIN\TRCSRVCE.EXE	Trace server
BIN\VMCCE.DLL	Access to connecting management for remote diagnostics
BIN\XREMOTCE.DLL	IFH for remote diagnostics
BIN\NETTCPCE.DLL	Communication DLL for remote diagnostics
BIN\NETPROCE.DLL	Communication DLL for remote diagnostics

4.1.1.4. Files for SCO-UNIX

The RUNTIME SYSTEM contains the following files:

readme	Installation notes
install	Installation program
uinstall	Uninstall program
profile	Profile extension
apiset	Script to set environment variables
/bin/apiboot	Start script for EDIABAS
/bin/apiclose	End script for EDIABAS
/bin/apitest	EDIABAS test program
/bin/bestinfo	Display BEST object file contents
/bin/devclose	Script to close the interface driver
/bin/pe	Password editor (PE)
/bin/printver	Output of EDIABAS version
/bin/ebasd	EDIABAS runtime system
/bin/ediabas	Program controlling communication software
/bin/ipc.ini	IPC configuration file
/bin/ipcctrl.que	Reference file
/bin/setlog	Script for controlling the installation messages
/bin/tracer	EDIABAS tracer
/bin/tracer.que	Reference file

4.1.1.5. Files for QNX

The RUNTIME SYSTEM contains the following files:

readme	Installation notes
install	Installation program
uinstall	Uninstall program
sysinit.add	Extension for system initialization file
ediabas.ini	EDIABAS configuration file
/bin/apiboot	Start script for EDIABAS
/bin/apiclose	End script for EDIABAS
/bin/apikill	Program to Stopp communication software
/bin/apisys	Program to start communication software
/bin/apisys.cfg	Configuration file of communication software
/bin/apitest	EDIABAS test program
/bin/apitrace	Trace of communication software
/bin/bestinfo	Display BEST object file contents
/bin/devclose	Script to close the interface driver

/bin/ediabas	EDIABAS runtime system
/bin/pe	Password editor (PE)
/bin/pinstall	Installation program
/bin/setlog	Script for controlling the installation messages
/bin/sigserv	Signal server
/ecu/tmode.prg	ECU description file for transparent mode

4.1.1.6. Documentation

The following documentation refers to the delivery package RUNTIME-SYSTEM (or to EDIABAS in general):

COMMENTS	Comments on this version (optional)
USER MANUAL	
INSTALLATION GUIDE	This document
ERROR REFERENCE	contains an overview on EDIABAS error messages
ECU SIMULATOR	Description of the ECU Simulator

4.1.2. Delivery package APPLICATION DEVELOPMENT KIT

The delivery package APPLICATION DEVELOPMENT Kit is available for all systems listed in section "Availability". This package allows application programs to be developed which use EDIABAS.

The following development systems are supported under MS-WINDOWS:

- Microsoft Visual C/C++ 6.0 (WIN32)
- Microsoft Visual C/C++ 1.52 (WIN16)
- Microsoft Visual Basic 6.0 (WIN32)
- Microsoft Visual Basic 4.0 (WIN32 and WIN32)

- Microsoft Visual Basic 3.0 (WIN16)

The following development systems are available for MS-WINDOWS CE:

- Microsoft eMbedded Visual C++ 4.2 SP2 + Pocket PC 2003 SDK (WINCE)
- Microsoft eMbedded Visual C++ 3.0 + Pocket PC 2002 SDK (WINCE)

4.1.2.1. Files for WIN32

The APPLICATION DEVELOPMENT KIT contains the following files:

README32.ADK	Installation notes
API\WIN32API.H	General C/C++ header files for all libraries
API\WIN32\APICALLS.C	C/C++ source code of APIW32.LIB and APIW32MT.LIB
API\WIN32\APIDLL.H	C/C++ header file for API32.DLL (Windows-DLL-Interface)
API\WIN32\VC60\APIW32.LIB	API-Library for single-Thread
API\WIN32\VC60\APIW32MT.LIB	API-Library for multi-Thread
API\ WIN32\VB60\API.BAS	Basic-Modul for Visual Basic 6.0
API\ WIN32\VB40\API.BAS	Basic-Modul for Visual Basic 4.0
BIN\APITST32.EXE	EDIABAS test application
ECU\TESTG.PRG	Example for group description file
ECU\TESTV.PRG	Example for variant description file

4.1.2.2. Files for WIN16

The APPLICATION DEVELOPMENT KIT contains the following files:

README.ADK	Installation notes
API\WIN16\API.H	General C/C++ header files for all libraries
API\WIN16\APICALLS.C	C/C++ source code of APIW.LIB
API\WIN16\APIDLL.H	C/C++ header file for API.DLL (WINDOWS-DLL-Interface)
API\WIN16\VC15\APIW.LIB	API-Library for Microsoft Visual C++ 1.52 (Large Model)
API\ WIN16\VB30\API.BAS	Basic-Modul for Visual Basic 3.0
API\ WIN16\VB40\API.BAS	Basic-Modul for Visual Basic 4.0
BIN\APITESTW.EXE	EDIABAS test application
ECU\TESTG.PRG	Example for group description file
ECU\TESTV.PRG	Example for variant description file

4.1.2.3. Dateien unter WINCE

The APPLICATION DEVELOPMENT KIT contains the following files:

API\WIN32API.H	General C/C++ header files for all libraries
API\\APICE.LIB	API-Library

4.1.2.4. Files for SCO-UNIX

The following files are contained on the delivery diskette APPLICATION DEVELOPMENT KIT and are copied to the target system.

readme	Installation notes
install	Installation program
/api/cc/api.c	Source file for producing libapi.a
/api/cc/apiipc.c	Source file for producing libapi.a
/api/cc/callback.c	Source file for producing libapi.a
/api/cc/errors.c	Source file for producing libapi.a
/api/cc/errortxt.c	Source file for producing libapi.a
/api/cc/help.c	Source file for producing libapi.a
/api/cc/job.c	Source file for producing libapi.a
/api/cc/jobdata.c	Source file for producing libapi.a
/api/cc/results.c	Source file for producing libapi.a
/api/cc/sco.c	Source file for producing libapi.a
/api/cc/trace.c	Source file for producing libapi.a
/api/include/api.h	INCLUDE file for producing libapi.a
/api/include/apimsg.h	INCLUDE file for producing libapi.a
/api/include/aspekte.h	INCLUDE file for producing libapi.a
/api/include/ bip.h	INCLUDE file for producing libapi.a
/api/include/callback.h	INCLUDE file for producing libapi.a
/api/include/config.h	INCLUDE file for producing libapi.a
/api/include/ediabas.h	INCLUDE file for producing libapi.a
/api/include/error.h	INCLUDE file for producing libapi.a
/api/include/help.h	INCLUDE file for producing libapi.a
/api/include/job.h	INCLUDE file for producing libapi.a
/api/include/jobdata.h	INCLUDE file for producing libapi.a
/api/include/kernel.h	INCLUDE file for producing libapi.a
/api/include/msgipc.h	INCLUDE file for producing libapi.a
/api/include/results.h	INCLUDE file for producing libapi.a
/api/include/sco.h	INCLUDE file for producing libapi.a
/api/include/sigserv.h	INCLUDE file for producing libapi.a
/api/include/trace.h	INCLUDE file for producing libapi.a
/api/include/traceapi.h	INCLUDE file for producing libapi.a
/api/include/typedef.h	INCLUDE file for producing libapi.a
/api/include/unixdef.h	INCLUDE file for producing libapi.a
/api/include/version.h	INCLUDE file for producing libapi.a
/api/lib/libapi.a	API library
/api/make/makefile	MAKE file for producing libapi.a
/api/obj/aspwrap.o	Object file for producing libapi.a
/api/obj/cfg.o	Object file for producing libapi.a

/api/obj/dynamic.o	Object file for producing libapi.a
/api/obj/log.o	Object file for producing libapi.a
/api/obj/process.o	Object file for producing libapi.a
/api/obj/profile.o	Object file for producing libapi.a
/api/obj/psig.o	Object file for producing libapi.a
/api/obj/scolib.o	Object file for producing libapi.a
/api/obj/svmsg.o	Object file for producing libapi.a
/api/obj/tracesco.o	Object file for producing libapi.a
/api/obj/watchdog.o	Object file for producing libapi.a

4.1.2.5. Files for QNX

The following files are contained on the delivery diskette APPLICATION-DEVELOPMENT-KIT and are copied to the target system.

readme	Installation notes
install	Installation program
/api/cc/api.c	Source file for producing libapi.a
/api/cc/apiipc.c	Source file for producing libapi.a
/api/cc/callback.c	Source file for producing libapi.a
/api/cc/errors.c	Source file for producing libapi.a
/api/cc/errortxt.c	Source file for producing libapi.a
/api/cc/help.c	Source file for producing libapi.a
/api/cc/job.c	Source file for producing libapi.a
/api/cc/jobdata.c	Source file for producing libapi.a
/api/cc/qnx.c	Source file for producing libapi.a
/api/cc/results.c	Source file for producing libapi.a
/api/cc/trace.c	Source file for producing libapi.a
/api/include/api.h	INCLUDE file for producing libapi.a
/api/include/apimsg.h	INCLUDE file for producing libapi.a
/api/include/aspekte.h	INCLUDE file for producing libapi.a
/api/include/ bip.h	INCLUDE file for producing libapi.a
/api/include/callback.h	INCLUDE file for producing libapi.a
/api/include/config.h	INCLUDE file for producing libapi.a
/api/include/ediabas.h	INCLUDE file for producing libapi.a
/api/include/error.h	INCLUDE file for producing libapi.a
/api/include/help.h	INCLUDE file for producing libapi.a
/api/include/job.h	INCLUDE file for producing libapi.a
/api/include/jobdata.h	INCLUDE file for producing libapi.a
/api/include/kernel.h	INCLUDE file for producing libapi.a
/api/include/msgipc.h	INCLUDE file for producing libapi.a
/api/include/qnx.h	INCLUDE file for producing libapi.a

/api/include/results.h	INCLUDE file for producing libapi.a
/api/include/sigserv.h	INCLUDE file for producing libapi.a
/api/include/trace.h	INCLUDE file for producing libapi.a
/api/include/traceapi.h	INCLUDE file for producing libapi.a
/api/include/typedef.h	INCLUDE file for producing libapi.a
/api/include/unixdef.h	INCLUDE file for producing libapi.a
/api/include/version.h	INCLUDE file for producing libapi.a
/api/lib/libapi.a	API library
/api/make/makefile	Make file for producing libapi.a
/api/obj/aspekte.o	Object file for producing libapi.a
/api/ecu/testg.prg	Test group description file
/api/ecu/testv.prg	Test variant description file

4.1.2.6. Documentation

The following documentation refers to the delivery package APPLICATION DEVELOPMENT KIT:

API USER MANUAL	Description of how the API libraries are used. Sample programs. Operation of the APITEST program.
API INTERFACE	Description of the API interface and how to operate it.
API FUNCTION REFERNCE	Description of the API functions
TRANSPARENT MODE	Description of the interface for transparent mode
INTERFACE DESCRIPTION	Transparent mode

4.1.3.Delivery package BEST DEVELOPMENT KIT

The delivery package BEST DEVELOPMENT KIT is only available for MS-WINDOWS. This package allows ECU description files to be developed.

The delivery package comprises, among other toolsBEST/2 Compiler, BEST/2 Debugger and BEST development environment.

The following files are contained on the delivery diskette BEST DEVELOPMENT KIT:

4.1.3.1. Files for WIN32

The delivery package BEST-DEVELOPMENT-KIT contains the following files:

README32.BDK	Installation notes
BIN\BEST2WIN.EXE	BEST/2 compiler
BIN\BEST2WIN.HLP	Help file for BEST/2 compiler
BIN\B2RUNTIM.LIB	BEST/2 runtime library
BIN\BESTVW32.EXE	BEST/2 debugger: BestView
BIN\BESTVW32.HLP	Help file for BEST/2 debugger
BIN\BESTBD32.EXE	BEST development environment: BestBoard
BIN\RUN1632.EXE	Help program for BestBoard
BIN\RUN16.EXE	Help program for BestBoard
BIN\APITST32.EXE	EDIABAS test application
BIN\APITAL32.EXE	EDIABAS test application
BIN\JOBLOO32.EXE	EDIABAS test application
ECU\TMODE.B1V	ECU description file for transparent mode (source code)
ECU\TESTG.B1G	Example group description file, BEST/1
ECU\TESTV.B2V	Example variant description file, BEST/2
TUTORIAL*.*	BEST-tutorial files

4.1.3.2. Files for WIN16

The delivery package BEST-DEVELOPMENT-KIT contains the following files:

README.BDK	Installation notes
BIN\BEST2WIN.EXE	BEST/2 compiler
BIN\BEST2WIN.HLP	Help file for BEST/2 compiler
BIN\B2RUNTIM.LIB	BEST/2 runtime library
BIN\BESTVIEW.EXE	BEST/2 debugger: BestView
BIN\BESTVIEW.HLP	Help file for BEST/2 debugger
BIN\BESTBRD.EXE	BEST development environment: BestBoard
BIN\APITESTW.EXE	EDIABAS test application
BIN\APITALKW.EXE	EDIABAS test application
BIN\JOBLOOPW.EXE	EDIABAS test application
ECU\TMODE.B1V	ECU description file for transparent mode (source code)
ECU\TESTG.B1G	Example group description file, BEST/1
ECU\TESTV.B2V	Example variant description file, BEST/2
TUTORIAL*.*	BEST-tutorial files

4.1.3.3. Documentation

The following documentation refers to the delivery package BEST-DEVELOPMENT-KIT:

BEST USER MANUAL	Programming description of a BEST/2 Description file. How to operate the supplied programs.
BEST/2 FUNCTION REFERENCE	Description of the BEST/2 function
BEST/2 LANGUAGE DESCRIPTION	Syntactic description of the BEST/2 language.

4.2. System requirements

4.2.1.MS-WINDOWS

In order to use EDIABAS under MS-WINDOWS, the following requirements must be meet:

IBM PC/AT or 100% compatible system with at least an 80386 processor

For EDIABAS RUNTIME SYSTEM WIN32 for MS-WINDOWS 95/98/ME a network installation is required.

For the remote function of the EDIABAS RUNTIME SYSTEM for WIN32 a network installation with TCP/IP is required.

4.2.2.MS-WINDOWS CE

In order to use EDIABAS under MS-WINDOWS CE, the system requirements has to be meet. A Pocket PC 2002/2003 compatible system is required.

For EDIABAS RUNTIME SYSTEM for WINCE a network installation of TCP/IP is required.

4.2.3.SCO-UNIX

In order to user EDIABAS under SCO-UNIX, a computer system which satisfies the following requirements is required:

IBM-PC or 100% compatible with at least a 80486 processor

- SCO OpenServer 5
- 12 MB RAM
- 3.5-in diskette drive (5.25-in on request)

4.2.4.QNX

A computer system which satisfies the following requirements is required in order to use EDIABAS under QNX:

IBM-PC or 100% compatible with 80386 processor or higher

- QNX 4.23
- 12 MB RAM
- 3.5" diskette drive

4.3. Installing for the first time

4.3.1. MS-WINDOWS

The installation manual for MS-WINDOWS is not part of this documentation.

4.3.2. SCO-UNIX

All shell scripts listed in this chapter require the kernel shell for execution.

IMPORTANT:
Pre-installed EDIABAS software version 1.3.0 or later can automatically be removed during the installation (optional). Older EDIABAS versions, however, must manually be deleted PRIOR TO installing the new EDIABAS RUNTIME SYSTEM (see also readme file). The step-by-step procedure is subsequently listed.

1. Log in as Root or Super-User

```
login root
```

2. Copy and install the (new) EDIABAS RUNTIME SYSTEMs

Calling 'install' (substitute the drive for '?')

```
mount /dev/fd?135ds18 /mnt
/mnt/install /mnt
umount /dev/fd?135ds18
```

Configure EDIABAS using the EDIABAS configuration file ediabas.ini (see EDIABAS configuration).

Example:

```
;EDIABAS sample configuration in ediabas.ini
[Configuration]
Interface = EDIC
EcuPath = /usr/ediabas/ecu
```

```
TracePath = /usr/ediabas
SimulationPath = /usr/ediabas/ecu
ApiTrace = 0
IfhTrace = 0
Simulation = 0
```

3. Copy and install the (new) APPLICATION DEVELOPMENT KITs

   Calling 'install' (substitute drive for '?')

```
mount /dev/fd?135ds18 /mnt
/mnt/install /mnt
umount /dev/fd?135ds18
```

4.3.3.QNX

1. Log in as either Root or Super-User

```
login root
```

2. Copy and install the EDIABAS RUNTIME SYSTEM

   After inserting the diskette "EDIABAS RUNTIME SYSTEM", invoke 'install' (substitute the drive for '?'):

```
mount /dev/fd? /mnt
/mnt/install /mnt
umount /mnt
```

You will be prompted during the installation whether EDIABAS is to automatically be started each time the system is booted. If EDIABAS is to beautomatically started, 'install'. Accordingly adapts the system initialization file sysinit.<node.

EDIABAS requires the message queue server 'Mqueue' from QNX. This must first be run BEFORE starting EDIABAS.

Example for starting 'Mqueue' in sysinit.<node>:

```
/bin/mqueue &
```

Configure EDIABAS via the configuration file /etc/ediabas.ini (see EDIABAS configuration).

   Example:

```
[Configuration]
EcuPath = /usr/ediabas/ecu
TracePath = /usr/ediabas
SimulationPath = /usr/ediabas/ecu
ApiTrace = 0
IfhTrace = 0
Simulation = 0
```

3. Copy and install the APPLICATION DEVELOPMENT KIT

After inserting the diskette "EDIABAS APPLICATION DEVELOPMENT KIT", invoke 'install' (substitute the drive for '?'):

```
mount /dev/fd? /mnt
/mnt/install /mnt
umount /mnt
```

4.4. EDIABAS program paths

4.4.1.MS-WINDOWS

The EDIABAS subdirectory BIN is to be entered in the search path of the system.

Example for AUTOEXEC.BAT:

```
SET OLDPATH=%PATH%
PATH=c:\ediabas\bin;%OLDPATH%
```

4.4.2.SCO-UNIX

The EDIABAS subdirectory "bin" is automatically entered in the search path.

4.4.3.QNX

The search path is left unchanged during installation. The complete path must be specified when calling EDIABAS programs.

4.5. EDIABAS configuration

The EDIABAS runtime behavior can be influenced via the EDIABAS configuration.

The EDIABAS configuration consists of information pairs which are listed in the area [Configuration] of file EDIABAS.INI. In this process, each line here describes a configuration element and its configuration setting:

Configurationelement = Configurationsetting

The configuration settings are read from file EDIABAS.INI when EDIABAS is first initialized. The default setting (see Table section 4.5.1) is assumed when configuration settings are missing or when file EDIABAS.INI does not exist.

All modifications of configurations in EDIABAS.INI must be made manually. NOTE: Modifications are only valid after EDIABAS has been re-started.

Certain configuration elements can also be modified at EDIABAS runtime by application programs. NOTE: These configuration changes only remain valid until EDIABAS is re-initialized.

Even ECU description files can modify configuration values during EDIABAS runtime. NOTE: These values are only valid, however, during the job.

4.5.1.Search sequence of EDIABAS.INI

Under MS-WINDOWS EDIABAS first searches for file EDIABAS.INI corresponding environment variable *EDIABAS_CONFIG_DIR*. If the environment variable was not set or file was not found in the directory, search is made in the Windows directory. If the file was not found in the windows directory, search was made in the EDIABAS program directory (subdirectory BIN).

Under SCO-UNIX and QNX EDIABAS first searches for file EDIABAS.INI corresponding environment variable *EDIABAS_CONFIG_DIR*. If the environment variable was not set or file was not found in the directory, search is made in the directory /etc. If the file was not found in the directory /etc, search was made in the EDIABAS program directory (subdirectory BIN).

4.5.2.Configuration overview

All configuration elements are listed below along with setting options and default values.

Each configuration element contains a description whether and how a modification is to be made (INI: via EDIABAS.INI, API: by application, BEST: by ECU description file).

All configuration elements in the following table are listed in the section **CONFIGURATION**.

Element	Note	Setting	Change	Default
ApiTrace	Control of the API Trace	**0** (OFF) **1** user trace **2** + time stamp **3** + process ID** **4** API trace **5** + time stamp **6** + runtime **7** + process ID** **8** debug trace	INI API ~~BEST~~	0
BipDebugLevel	Debug level in ECU description files	**0** .. **32767**	INI API ~~BEST~~	0
EcuPath	Path of the ECU description files	*Path*	INI API ~~BEST~~	.
EdiabasIniPath	Path of the configuration file EDIABAS.INI used (only if this exists)	*Path*	~~INI~~ ~~API~~ ~~BEST~~	
EdiabasVersion	EDIABAS version	*String*	~~INI~~ ~~API~~ ~~BEST~~	
IfhTrace	Control of IFH Trace	**0** (OFF) .. **1** User interface **2** + interface **3** + timestamp	INI API BEST	0

IfhnTrace	Control of IFH-network-trace. Only available with XREMOTE.	**0** (aus) **1** User Interface **2** + Interface **3** + timestamp	INI ~~API BEST~~	0
IfhVersion	IFH version	*String*	~~INI~~ API ~~BEST~~	
Interface	Hardware interface	*String*	INI ~~API~~ ~~BEST~~	EDIC
IgnitionHandling	Ignition ON/OFF handle as error	**0** (OFF) .. **1** (ON)	INI API BEST	1
LoadWin32	Selection of EDIABAS-systems for WIN16-applications	**0** (EDIABAS for WIN16) .. **1** (EDIABAS for WIN32)	INI ~~API~~ ~~BEST~~	Windows 9x/ME: 0 Windows NT/2000/ XP: 1
NetworkProtocol	Select network protocol for the remote diagnostics	**TCP**	INI ~~API~~ ~~BEST~~	-
RetryComm	Repetition on communication error	**0** (OFF) .. **1** (ON)	INI API ~~BEST~~	1
Simulation	Control of ECU simulation	**0** (OFF) .. **1** (ON)	INI API ~~BEST~~	0
SimulationPath	Path of the ECU simulation files	*Path*	INI API ~~BEST~~	.
SystemResults	Control whether to store: ignition, supply voltage and job status - system results	**0** (do not store) .. **1** (store)	INI API ~~BEST~~	1
TaskPriority	Priority setting for EDIABAS (0 = optimal setting, 1 = lowest priority)	*0 - 10*	INI API ~~BEST~~	0
TracePath	Path of the Trace files	*Path*	INI ~~API~~ ~~BEST~~	.
TraceSize	max. trace file size in KB	**0** .. **32767**	INI ~~API~~ ~~BEST~~	1024
CodeMapping	file path of the code page (only with code conversion)	*path*	INI ~~API~~ ~~BEST~~	.
UbattHandling	Supply voltage ON/OFF handle as error	**0** (OFF) .. **1** (ON)	INI API BEST	1

** Note: the process ID is only indicated with EDIABAS for WIN32.

The configuration elements for the network protocol TCP (section **TCP**) are listed in the following table. This section is only required when the value TCP has been assigned to the configuration element **NetworkProtocol** in section **CONFIGURATION**.

Element	Note	Setting	Change	Default
Port	Port number	1000...30000	INI ~~API~~ ~~BEST~~	-
RemoteHost	Name or IP address of remote system	name/addresse	INI ~~API~~ ~~BEST~~	-
TimeoutConnect	timeout in milliseconds for connection setup	1000 ... 59000	INI ~~API~~ ~~BEST~~	5000
TimeoutReceive	timeout in milliseconds for receipt of a message	1000 ... 59000	INI ~~API~~ ~~BEST~~	5000
TimeResponsePending	Time in milliseconds Zeit in Millisekunden between alive messages. Not supported with EDIABAS V6.4.0	500 ... 59000	INI ~~API~~ ~~BEST~~	2000
TimeoutFunction	Timeout in milliseconds for a for a long IFH function (for example send_and_receive). The complete timeout is: **TimeoutFunction** + **TimeoutReceive**.	5000 ... 59000	INI ~~API~~ ~~BEST~~	59000
RemoteHost	Name or IP address of the computer to be controlled remotely	Name/address	INI ~~API~~ ~~BEST~~	-

4.5.3.Select the hardware interface

4.5.4.MS-WINDOWS

The hardware interface has to be set in the configuration file EDIABAS.INI (**Interface**).

The following hardware interfaces are currently supported:

- SOFTING EDIC

The corresponding interface driver is NOT a constituent of the delivery package RUNTIME SYSTEM.

4.5.5.SCO-UNIX

In the SCO-UNIX version the hardware interface driver cannot be set (The corresponding setting **Interface** in EDIABAS.INI is ignored). It is not part of the RUNTIME SYSTEM.

4.5.6.QNX

In the QNX version the hardware interface driver cannot be set (The corresponding setting **Interface** in EDIABAS.INI is ignored). It is not part of the RUNTIME SYSTEM.

4.5.7.Directory of ECU description files

The directory of the ECU description files (referred to as SGBDs) to be used is to be entered manually in EDIABAS.INI (**EcuPath**).

For further information, read the section "Administration".

4.5.8.Character Set Conversion

The representation of characters not defined in the 7-bit ASCII character set (such as ö or ä) depends on the code table used by the operating system. If SGBDs are developed on a computer different from the target computer and the computers do not use the same code table, the representation of the EDIABAS result string may differ in the two computers.

This problem can be avoided with the EDIABAS configuration **CodeMapping**. A code page to be defined with this configuration (file with 256 characters) is loaded in the program memory when EDIABAS is started and causes all EDIABAS result strings to be automatically converted to the desired character format.

The code page ANSI2OEM.TAB, which is part of the delivery, converts the ANSI (Windows) character set into the OEM (DOS) character set.

For the conversion, each character of an EDIABAS result string is replaced by the corresponding character from the code page. For conversion, the character to be replaced is used as an index which points to the new character on the code page.

Example:

ANSI Character Set (Windows)	OEM Character Set (DOS)
`G (0x47)`	`G (0x47)`
`E (0x45)`	`E (0x45)`
`R (0x52)`	`R (0x52)`
`Ä (0xC4)`	`Ä (0x8E)`
`T (0x54)`	`T (0x54)`

For converting, for example, the character ‘Ä‘ from the ANSI character set (Windows) to the OEM character set (DOS), the code 0xC4 must be replaced by 0x8E.

For characters which have no counterpart in the other code, "." is indicated.

4.5.9.WIN16/32-Umsetzung

For the operating systems Windows 9x/ME/NT4/2000/XP an automatic conversionof EDIABAS/WIN16 calls to EDIABAS/WIN32 is possible.

- LoadWin32 = 0

  Using EDIABAS/WIN16.

- LoadWin32 = 1

  Using EDIABAS/WIN32 with conversion of all API calls.

4.6. Start and terminate EDIABAS

4.6.1.WIN32

When an application with access on EDIABAS is started the following components are loaded automatically: API32.DLL, EBAS32.DLL, EBAS32.EXE and TRACEX32.EXE. EBAS32.DLL loads the interface handler.

Access of applications to EDIABAS takes place via API32.DLL.

4.6.2.WIN16

The first time an application accesses EDIABAS, EDIABASW.EXE is automatically started.

EDIABASW.EXE subsequently loads the configured Interface Handler.

EDIABASW.EXE is accessed by applications via API.DLL.

4.6.3.WINCE

When an application with access to EDIABAS is started, the EDIABAS components APICE.DLL, EBASCE.DLL and TRCSRVCE.EXE are automatically loaded. During the first access to EDIABAS, EBASCE.DLL loads the configured Interface Handler.

Access of applications to EDIABAS takes place via APICE.DLL.

4.6.4.SCO-UNIX

After installing EDIABAS (see "Installing from diskette for the first time"), shut down and re-boot the system.

After installation, EDIABAS will automatically be booted and shut down by the system. The EDIABAS configuration file ediabas.ini is only read once the first time an application program is accessed.

EDIABAS is accessed by applications via the library libapi.a. which is bounded to the application program.

Normally, EDIABAS is started up or shut down automatically during the corresponding system processes. In addition, EDIABAS can be manually started up or shut down with Super-User authorization.

Manual EDIABAS startup (if EDIABAS does not run yet):

```
/usr/ediabas/bin/apiboot
```

Manual EDIABAS shutdown (if EDIABAS is already running):

```
/usr/ediabas/bin/apiclose
```

4.6.5.QNX

If EDIABAS was selected to automatically start during the installation, the system must be re-started.

Otherwise, EDIABAS must be manually started (with Super-User access rights):

```
/usr/ediabas/bin/apiboot
```

The EDIABAS configuraiton file ediabas.ini is read only once the first time the application is accessed.

Applications access EDIABAS via the library libapi.a.linked to the application program.

Normally, EDIABAS must not be terminated. If, however, this is necessary, the instruciton below can be used:

```
/usr/ediabas/bin/apiclose
```

4.7. Un-install

When un-installed, all EDIABAS delivery packages are removed from the system. Subdirectory ECU (or ecu under SCO-UNIX) containing the ECU description files is only deleted when this is requested by the user. The subdirectory ECU and the residing ECU description files are generally not deleted.

4.7.1.MS-WINDOWS

The Uninstall manual is not part of this documentation.

4.7.2.MS-Windows CE

The Uninstall manual is not part of this documentation.

4.7.3.SCO-UNIX

EDIABAS software can be un-installed using the program UINSTALL. All programs which access EDIABAS must be terminated prior to un-installing software. EDIABAS software is to be removed in the following way:

1. Log in as Root or Super-User

```
login root
```

2. Stop any active EDIABAS software and delete the EDIABAS software using uinstall

```
/usr/ediabas/uinstall
```

The un-install program uinstall allows EDIABAS software to be removed without removing the ECU description files (optional).

WARNING:
Softing-external software in the directory /usr/ediabas will be deleted! Always assure in advance that this is stored in another directory.

3. Delete all API or EDIABAS entries in /etc/profile.

4.7.4.QNX

The EDIABAS software can be un-installed using the program UINSTALL. Before un-installing the software, all programs which access EDIABAS must be terminated first. The EDIABAS software is to be un-installed again in the following manner:

1. Log in as Root or Super-User

```
login root
```

2. Stop any active EDIABAS software, and delete EDIABAS software using uinstall

```
/usr/ediabas/uinstall
```

IMPORTANT: First assure that no Softing-external software resides in the directory /usr/ediabas, since this will also be deleted!

4.8. Protection mechanism

EDIABAS accesses ECU description files (SGBDs) for processing API jobs (see chapter 3). ECU description files are developed as source code (B2V-, or B2G file) in BEST/2 and subsequently compiled into an object format (PRG-, or GRP file) using the BEST compiler for use with EDIABAS. The object files are loaded and executed at runtime by the runtime system.

In order to protect ECU description files in object format, EDIABAS offers a protection mechanism. This mechanism not only prevents unauthorized reading of file contents (job names, table names, table contents, etc.) but also prevents unauthorized use by users (non-authorized external companies or private persons). The ECU description files are protected in two ways: 1) File information is coded and not written in plain-language text and 2) the file cannot run under a non-authorized EDIABAS runtime system; i.e. it cannot be loaded by EDIABAS. Thus, information cannot be read from the ECU description files, and jobs within cannot be executed.

After the installation, the EDIABAS runtime system first assumes a state in which only unprotected ECU description files can be executed. ECU description files are protected by means of passwords specified for the runtime system and for the ECU description files. Passwords for the runtime system are specified using the Password Editor (PE) (see section 5.3), whereas passwords in the ECU description files are entered using the BEST compiler BEST2WIN (see BEST User Manual sections 5.1 and 5.2).

NOTE: Only the programs PE and BEST2WIN included in the current EDIABAS installation can be used for specifying passwords.

To protect an EDIABAS system, perform the following two steps:

1. In order to protect the EDIABAS runtime system, system passwords must be defined (at least one) using the Password Editor (see section 5.3).

2. Next, all ECU description files used must be compiled with at least one password from the list of the passwords entered above (see BEST user Manual, sections 5.1 and 5.2).

At runtime, the data read when loading the ECU description file must first be decoded. Afterwards, the runtime system checks whether one of the passwords from the ECU description file is known by the runtime system. If the runtime system recognizes at least one password from the ECU description file, the ECU description file can be executed. The EDU description files which are not protected with a password can be executed from every EDIABAS runtime system (protected or not) beginning with version 5.5

4.9. Remote diagnostics

EDIABAS for WIN32/CE allows access to diagnostic interfaces and, consequently, cabled ECUs which are connected to another PC.

This requires a network connection of local and remote-controlled PC via TCP/IP as well as a WIN32 operating system supported by EDIABAS.

The application and EDIABAS run on the local PC, whereas the Interface Handler (IFH) as well as the IFH server run on the remote-controlled PC. Before the remote-controlled PC can be accessed, the IFH server IFHSRV32.EXE must be started first.

The remote control is enabled or controlled via the EDIABAS configuration file EDIABAS.INI. EDIABAS is configured manually on both PCs.

The EDIABAS configuration parameters **TracePath** and **SimulationPath** are not transmitted to the remote-controlled PC from the local PC. Instead, the corresponding configuration parameters of file EDIABAS.INI (contained on the remote-controlled PC) are used.

The network protocol TCP (entry **NetworkProtocol**) as well as a user-selectable port number is to be specified on both PCs. The port number must be identical on both PCs and must **not** collide with the other TCP applications (1000 < port number < 30000).

4.9.1.EDIABAS.INI on the local PC

The configuration file EDIABAS. INI must be edited on the local PC in accordance with the list depicted below. The assignment REMOTE to the configuration element **Interface** causes remote control of the PC, which is specified with the entry **RemoteHost**. Specification of a logical computer name requires a corresponding HOSTS file or a DNS service.

```
[Configuration]
Interface = REMOTE
NetworkProtocol = TCP
```

```
[TCP]
RemoteHost = <Name/addresses of remote-controlled PC>
Port = <Port from remote-controlled IFH>
TimeoutConnect=<Timeout for connection setup>
TimeoutReceive=<Timeout for Receive>
TimeoutFunction=<Timeout for IFH >
```

Example of EDIABAS.INI for the local PC:

```
[Configuration]
Interface=REMOTE
NetworkProtocol=TCP

[TCP]
RemoteHost=193.29.29.175
Port=5000
TimeoutConnect=2000
TimeoutReceive=2000
TimeoutFunction=50000
```

4.9.2.EDIABAS.INI on the remote-controlled PC

Configuration file EDIABAS.INI must be extended on the remote-controlled PC in accordance with the list depicted below.

```
[Configuration]
Interface=<Interface>
NetworkProtocol=TCP

[TCP]
Port=<Port of the remote-controlled IFH>
```

Example of EDIABAS.INI for the remote-controlled PC:

```
[Configuration]
Interface=EDIC
NetworkProtocol=TCP

[TCP]
Port=5000
```

4.9.3.Procedure for remote diagnostics

1. Install EDIABAS on the local or remote-controlled PC.

2. Install the EDIABAS configuration on the local and remote-controlled PC.

3. Start the IFH server on the remote-controlled PC.

4. Start the application on the local PC.

4.10. EDIABAS Parallel Operation

EDIABAS for WIN32/CE and SCO-UNIX enable different application programs to access EDIABAS at the same time.

In contrast to the other EDIABAS platforms, WIN32/CE and SCO-UNIX has no single one EDIABAS which runs all application programs consecutively. Instead, a separate EDIABAS runs for each application program and if nec. is started (and under WIN32/CE) finished automatically.

Parallel operation of several interfaces of the same type is only possible if this is supported by the corresponding interface software.

Under SCO-UNIX up to 64 application programs may access on EDIABAS at the same time.

5. Help programs

5.1. UINSTALL

Remove the EDIABAS software.

This program is only available under SCO-UNIX and QNX. UINSTALL deletes all installed EDIABAS delivery packages. The subdirectory ECU (or ecu under SCO-UNIX and QNX) containing the ECU description files is not deleted.

5.1.1.SCO-UNIX

This program can only be called by the Superuser. UINSTALL deletes all programs in the EDIABAS directory (**also EDIABAS-external software!**). After executing the program, all API or API or EDIABAS entries should be deleted in /etc/profile .

Call:

/usr/ediabas/uinstall

Example:

```
login root
/usr/ediabas/uinstall
```

5.1.2.QNX

Only the Superuser can invoke this program. UINSTALL can be used to delete all programs in the EDIABAS directory (also EDIABAS-external software).

Call:

/usr/ediabas/uinstall

Example:

```
login root
```

```
/usr/ediabas/uinstall
```

5.2. PE (Password Editor)

The Password Editor (PE) manages the passwords of a protected EDIABAS version. The Password Editor is called PE.EXE. A password entry consists of a public label and associated, secret password. The password is only used once in the entry. Only the password label is then used for all further accesses.

A maximum of 10 label/passwords can be entered.

Labels/passwords which have once been entered cannot be deleted anymore. If this is desired, EDIABAS must be re-installed.

Each label/password can be disabled or enabled (more than once possible). A disabled label/password behaves during runtime as if it did not exist.

A `<Label>` consists of 1 to 10 ASCII characters in ANSI code, where only ASCII characters are permitted in the range: `20h <= <ASCII-character> <= 7Eh`. The character '@' (40h) is illegal.

A `<Password>` consists of 6 to 10 ASCII characters in ANSI code, where only ASCII characters are permitted in the range: `20h <= <ASCII characters> <= 7Eh`.

The Password Editor is controlled via the command line. It offers the following functionalities:

- **Display all commands of the Password Editor**
  This Password Editor function represents a help function and displays all available commands.
  Call: `pe`
- **View all labels of the password file**
  With the Password Editor, the labels of all passwords which are known by the runtime system can be displayed along with their status (enabled/disabled).
  Call: `pe -v`
- **Add a new entry in the password file**
  Enter a new label/password in the EDIABAS runtime system. A total of 10 entries are possible. When entries are added, they automatically are assigned the status "enabled".
  Call: `pe -a <Label>=<Password>`
- **Enable an entry in the password file**
  Disabled entries can be re-enabled again. Possible more than one.
  Call: `pe -e <Label>`

- **Disable an entry in the password file**
  Entries which have been entered can be disabled. In ECU description files sequence, disabled entries behave as entries which do not exist. Possible more than once.
  Call: `pe -d <Label>`
- **Check whether an ECU description file on the installed on the EDIABAS system can be executed**
  The Password Editor can also be used to check whether ECU description files can be executed in connection with the installed EDIABAS system. For this purpose, the password editor fetches the passwords from the ECU description file and compares them to the system passwords.
  Call: `pe -c <SGBD>`

The Password Editor returns the value 1 in the case of an error, otherwise 0.

NOTE: Only the Password Editor included in the EDIABAS installation can be used.

5.3. DEVCLOSE

Close the interface driver

This program is only available under SCO-UNIX and QNX. If, after using EDIABAS, an EDIABAS-external software wants to access the interface driver and this is still in the open state, the interface driver can be closed with the program DEVCLOSE.

Call:

```
devclose
```

5.4. BESTINFO

Display of ECU description file contents

The program BESTINFO allows the output of the following information regarding the specified ECU description file (object format):
- File name
- Revision number
- Last user (operator) (output of maximum 63 characters), date of the last change
- List of all jobs

Call:

bestinfo *sgbd*

Example:

```
bestinfo testv.prg
```

Produces output (example) :

```
BEST object file :  testv.prg
BIP version      :  03.03.00.00
Revision number  :  2.0

Last modification by softing.sag.Os , Thu Mar 25 16:33:10
1993

5 Jobs:
        INITIALISIERUNG
        IDENT
        CHECK
        BINPARA
        WAIT

BESTINFO ready.
```

5.5. BESTVER

Version test of ECU description files

This program is only available under MS-WINDOWS. It allows the versions of ECU description files (object format) to be tested. In addition it allows specification of a uniform version number of the package for several description files.

Call:

BESTVER[**-R** *RevMAJ.RevMIN username*] [**-V** *Packageversion*] *sgbd(en)*

Example:

```
bestver -R 1.6 Softing -V 20 testv.prg
```

Modifies the revision (revision 2.0 becomes 1.6, and author becomes Softing) and specifies a package version number (20). The following is output:

```
TESTV.PRG
         BIP version      : 03.03.00.00
         Revision number  : 2.0
         Last change      : Thu Mar 25 16:33:10 1993
         By               : softing.sag.Os
         Package version  : 00000000
 ==>
         Revision number  : 1.6
         Last change      : Wed Nov 24 12:41:50 1993
         By               : Softing
         Package version  : 00000020
```

Output of the last user is limited to a maximum of 63 characters.

5.6. XTRACT

Display help texts from ECU description files

This program is only available under MS-WINDOWS. XTRACT can be used to display the help texts stored in the BEST object files.

Call:

> XTRACT [**-F**] *objectfile(s)* [*outdir*]
> XTRACT -? (shows an overview of options)

Option **-F** pipes the output is a file with the extension **.biv** or **.big**. The directory can be specified where the output file is to be produced. Wildcards can be used in *objectfile*.

Example:

```
xtract -F testv.prg
```

Produced file testv.biv containing the following:

```
ECU:TESTV
ORIGIN:softing.sag.Os
REVISION:2.0
AUTHOR:softing.sag.Os
ECUCOMMENT:Description file WIHTOUT access to the interface
ECUCOMMENT:Orignal: TESTV.B1V
JOBNAME:initialization
JOBCOMMENT:Initialization
RESULT:DONE
RESULTTYPE:int
RESULTCOMMENT:1 if ok, otherwise 0
JOBNAME:IDENT
JOBCOMMENT:Determine the identification string
RESULT:JOB_STATUS
RESULTTYPE:string
RESULTCOMMENT:Returns: OKAY or ERROR_PARAMETER
...
RESULTCOMMENT:Returns: OKAY or ERROR_PARAMETER
```

Automatic table output:

For improved documentation of BEST objects, XTRACT also offers the functionality to output the entries of a table instead of a BEST/2 comment. For this purpose, a BEST/2 comment must have the following structure:

```
comment: table <Table name> <Column> [ <Column> ... ]
```

Example of a BEST/2 source file:

```
...
table beispielTabelle[2][]=
{
{ "COLUMN1",    "COLUMN2",     "COLUMN3"   },

{ "Line1",      "xtract",      "generated" },
{ "Line2",      "these",       "texts"     },
{ "Line3",      "automatically",    "!"         }
};

job ( name = beispielJob;
      comment : Start of comment
      comment : Table beispielTabelle SPALTE2 SPALTE3
      comment : End of comment
...
```

XTRACT outputs the following lines in the example above:

```
...
JOBNAME:beispielJob
JOBCOMMENT:Start of comment
JOBCOMMENT: "xtract" "generated"
JOBCOMMENT: "these" "Text"
JOBCOMMENT: "automatically" "!"
JOBCOMMENT:End of comment
...
```

5.7. STRIP

Remove (strip) the extra information from ECU description files

This program is only available under MS-WINDOWS. STRIP can be used to remove the help text and debug information stored in BEST object files.

Call:

> STRIP [**-N**] **dh** *sourcefile(s)* [*outdir*]
> STRIP -? (Shows an overview of options)

Parameter **d** indicates that the debug information is to be removed, whereas parameter **h** indicates that the help texts are to be removed. The directory can be specified where the output file is to be produced. If an object file is overwritten which already exits, STRIP requests the user for re-confirmation. This interrogation can be suppressed with option **-N**.

Example:

```
    strip -N dh testv.prg
```

Remove the debug information and help texts from file testv.prg.

A. GLOSSARY

API (**A**pplication **P**rogramming **I**nterface):
EDIABAS interface up to the application.

Application:
Application program for executing the diagnostics and coding ECUs.

BEST (**BE**schreibungssprache für **ST**euergeräte):
Description language for ECUs.

Description file:
Contains all data and methods which are required for describing the diagnostic function of an ECU.

Device:
Logical device used for connecting the interface.

Diagnostic concept:
Conventions according to ISO in which access to the diagnostic data is described in ECUs.

DLL (**D**ynamic **L**ink **L**ibrary):
Dynamic function library under MS-Windows. Linked at the to the application only at runtime.

ECM (**E**lectronic **C**ontrol **M**odule):
Abbreviation for ECU.

ECU (**E**lectronic **C**ontrol **U**nit):
Abbreviation.

EDIABAS (Elektronik **DIA**gnose **BAS**issystem):
Electronic Diagnostic Basic System. API, runtime system and Interface Handler.

EDIC (Enhanced **D**iagnostic **I**nterface **C**omputer)

IFH (Inter**F**ace **H**andler):
Software interface for operating the EDIC functions. Communication with the EDIC interface occurs via the EDIC driver.

Job:
Method in the ECU description file which reads data from the ECU.

Runtime system:
EDIABAS runtime system with BEST interpreter.
The runtime system processes a job issued via API by loading and interpreting the corresponding ECU description file. In addition to the ECU-specific parameters, these ECU description files contain the corresponding rules, methods and sequences for processing jobs.

SGBD (**S**teuer**G**eräte**B**eschreibungs**D**atei):
ECU (Electronic Control Unit) description file.

SG (**S**teuer**G**eräte):
See ECU (Electronic Control Unit).

Control device variant:
Functionally different versions of ECU groups. No difference is made in version type (e.g., country version, coding variant).
The ECU is also to be conceived as an ECU variant for only one version within an ECU group.