EDIABAS

Electronic Diagnostic Basic System

BEST USER MANUAL

VERSION 6d

BESTUSER.DOC

CONTENTS

1. Revision history

Version 3.0	First release
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 by appendix "Limits and restrictions"
Version 6	Extension for EDIABAS V6.0.0
Version 6a	Extension of appendix “Limits and restrictions”
Version 6d	Revised for EDIABAS V6.4.4

2. Introduction

2.1. About this Manual

This manual sets out the principles for creating control unit description files and describes the use and options of the development support programs. This manual does not go into any great depth about BEST/1 language. As a general rule all control unit description files can be formulated in BEST/2 notation which is easier to read. You will find a detailed description of the BEST/1 language in [3]. General information about EDIABAS and control unit description files is given in Reference [2].

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".

2.4. Trademarks

Microsoft, MS, MS-DOS, Windows and WIN32 are registered trademarks of the Microsoft Corporation.

3. General Information

3.1. Job concept

Control unit description files - SGBDs for short - contain methods for reading control unit specific data from the control unit and compiling it into an abstract and generally readable form.

In EDIABAS, these methods are called jobs and are usually written in BEST/2. BEST/2 syntax is based on C language but is matched to the specific needs of control unit description. You will find a detailed description of BEST/2 syntax in [5]. In particular, the jobs have had to be incorporated as a special type of function. The use of fields has also been streamlined to simplify the handling of complex telegram data. Unlike C, the user cannot define his own functions, but there is a runtime library (see [1]) that has been optimized for diagnostic tasks and it provides functions for all problems.

A job can be called independently of other jobs at any time so jobs are not allowed to be dependent on one other in time terms. BEST/2 prevents this by not having any constructs that allow jobs to exchange data. Only constants may be global.

3.2. ECU description files (SGBDs)

There are three types of ECU description files: 1) Variant, 2) Basic and 3) Group. An explanation of the terms "ECU variant" and "ECU group" can be found in [4]).

EDIABAS expects all ECU description files in the ECU directory (configuration element **EcuPath** in EDIABAS.INI).

3.2.1. Group description files

In addition to the jobs INITIALISIERUNG and ENDE (optional), the group description file only contains the job IDENTIFIKATION which determines the

variant currently installed in the vehicle. Jobs in a group description file cannot be called with an API job.

3.2.2. Variant description files

In addition to the jobs INITIALISIERUNG and ENDE (optional), the variant description files contain the actual jobs for determining data from the ECU.

A variant description file can access one or more basic description files. Jobs in the basic description files are treated like the jobs from the variant description file when jobs are processed.. Standard jobs in the basic description file are ignored.

3.2.3. Basic description file

Jobs which are the same for several ECUs are stored in commonly used basic ECU description files (SGBDs). ECU-specific jobs, in contrast, are stored in the respective "variant" ECU description file.

In order not to limit the alternative search to the single basic ECU description file, alternative basic ECU description files can be specified within the ECU description file.

3.2.3.1. Access to jobs in a basic ECU description file

Which basic ECU description file or alternative ECU description files are to be applied whenever the job in the ECU description file is not found can be specified in the header of a variant ECU description file. For this purpose, the new entry **uses** is defined in the header followed by the name(s) (without extension) of the alternative basic ECU description files (PRG files), separated by one or more blanks and/or comma. The line must end with a semicolon.

Example:

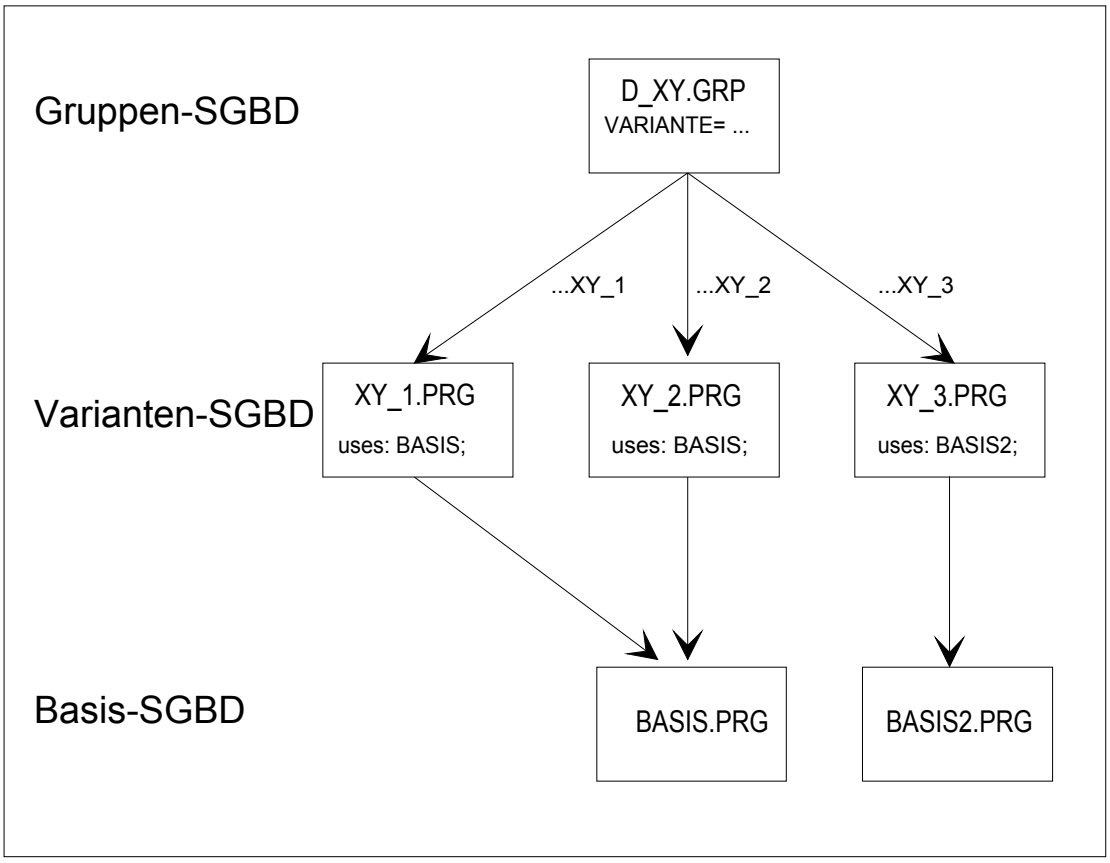
The example below shows the header of a variant ECU description file designated VARIANTE.PRG. Jobs which are not found in this ECU description file should be searched in the basic ECU description files BASIS1, BASIS2 and BASIS3. This is defined in the line **uses**.

```
/////////////////////////////////////////////////////////////
//        H E A D E R
/////////////////////////////////////////////////////////////

ecu     : Demo ECU VARIANTE.PRG;
origin  : softing.BG5.Ma;
revision: 1.0;
author  : softing.BG5.Ma;
language: German;
uses    : BASIS1, BASIS2, BASIS3;
comment : Example for using basic ECU description files (SGBDs);

/////////////////////////////////////////////////////////////
```

EDIABAS first checks whether the job specified in the **apiJob** call can be found in VARIANTE.PRG. If this is the case, the job is executed. If the job does not exist in VARIANTE.PRG, search is continued in the first alternative basic ECU description file specified in the header from VARIANTE.PRG following **uses**, here BASIS1.PRG. Search is continued accordingly until the job is found. If the job is also not found in the last ECU description file (here BASIS.PRG), EDIABAS issues the error message "SYS-0008: JOB NOT FOUND" .

In another example, assumption is made that the application in the **apiJob** call specifies the name of a group ECU description file (here D XY) beside the job name. The ECU variant is determined in the job identification of the group ECU description file D_XY.GRP , and the associated name of the variant ECU description file is entered in the result **VARIANTE**: In this case, XY_1 or XY_2 or XY_3. EDIABAS searches the job in the corresponding ECU description file. Jobs which are not found there are searched in the basic ECU description file (here BASIS.PRG or BASIS2.PRG). The reference to the basic ECU description file must always be entered in the variant ECU description files.

A variant ECU description file must exist for each ECU variant which, in the extreme case, only consists of the header and the jobs INITIALISIERUNG and ENDE (optional). When a basic ECU description file is accessed, the standard jobs INITIALISIERUNG and ENDE of the basic ECU description file are not called (the standard jobs INITIALISIERUNG and ENDE are not required in a basic ECU description file).



Access to basic ECU description file

3.2.3.2. "Overloading" a job from the basic ECU description file

If a job is to be an ECU-specific variant and not to be called from the basic ECU description file, this job is to be appropriately defined in the variant ECU description file. For this purpose, the corresponding job is stored in the variant ECU description file of the ECU under the same name and implemented ECU specifically. Since EDIABAS already finds the job in the variant ECU description file, this is executed inste4ad of the job with the same name from the basic ECU description file.

3.3. Standard jobs

Standard jobs have a special meaning for EDIABAS. These jobs are automatically called by the runtime system under certain conditions in addition to the job specified. These jobs can receive parameters.

The result of standard jobs is stored in defined standard results which EDIABAS automatically interrogates.

A job which is issued via a variant is processed as shown below:

Application program Calls	**EDIABAS** Selects SGBD:	**EDIABAS** Processes jobs:
apiInit()		
apiJob(SGBD1,JOB1,...)		
	SGBD1.PRG	INITIALISERUNG JOB1
apiJob(SGBD1,JOB1,...)		
	SGBD1.PRG	JOB1
apiJob(SGBD1,JOB1,...)		
	SGBD1.PRG	JOB1 (error occurred)
apiJob(SGBD1,JOB1,...)		
	SGBD1.PRG	INITIALISERUNG JOB1
apiJob(SGBD2,JOB1,...)		
	SGBD1.PRG	ENDE
	SGBD2.PRG	INITIALISIERUNG JOB1
apiInit()		
apiJob(SGBD3,JOB1,...)		
	SGBD2.PRG	ENDE
	SGBD3.PRG	INITIALISIERUNG JOB1

A job which is issued via a group is processed as shown below:

Application program Calls	**EDIABAS** Selects SGBD:	**EDIABAS** Processes jobs:
apiInit()		
apiJob(GRUPPE,JOB1,...)		
	GRUPPE.GRP	INITIALISERUNG IDENTIFIKATION (of the variant) ENDE
	VARIANTE.PRG	INITIALISIERUNG JOB1
apiJob(GRUPPE,JOB1,...)		
	VARIANTE.PRG	ENDE
	GRUPPE.GRP	INITIALISERUNG IDENTIFIKATION (of the variant) ENDE
	VARIANTE.PRG	INITIALISIERUNG JOB1
apiInit()		
apiJob(GRUPPE,JOB1,...)		
	GRUPPE.GRP	INITIALISERUNG IDENTIFIKATION (of the variant) ENDE
	VARIANTE.PRG	INITIALISIERUNG JOB1

3.3.1. Standard job INITIALISIERUNG

The standard job INITIALISIERUNG must exist in every group and variant description file. It is automatically called:

- After previous initialization of API (**apiInit()/apiInitExt**)
- After previous device switch (**apiSwitchDevice()**)
- After previous occurrence of an error
- For a job with pre-specified variant (only after changing the ECU variant)
- For a job whose variant is unknown

Successful initialization is to be displayed with the standard result DONE (result type **int** or **unsigned int**):

- Result DONE <> 0 Successful initialization
- Result DONE = 0 Erroneous initialization
- Result DONE does not exist Erroneous initialization

Except for the result DONE, the standard job INITIALISIERUNG must not return any other results .

3.3.2. Standard job IDENTIFIKATION

The standard job IDENTIFIKATION is called to determine the installed ECU variant whenever a group description file has been specified in the job.

The standard job IDENTIFIKATION must exist in every group description file.

An identified ECU variant is to be displayed in the first result record with the standard result VARIANTE (string from result type **string**):

- Result VARIANTE = String (at least 1 character) Successful identification, the result represents the name of the ECU variant
- Result VARIANTE = Blank string Erroneous identification
- Result VARIANTE Does not exist Erroneous identification

Except for VARIANTE, the standard job IDENTIFIKATION must not delivery any other results.

3.3.3. Standard job ENDE

The optional standard job ENDE is called when the description file is varied:

- Before changing the description file
- Before a job with unknown variant

The standard job ENDE can exist in each group and variant description file.

The standard job ENDE must not return results.

3.4. Variant identification

The variant is identified by the job IDENTIFIKATION which is mandatory in the group description file. This job must return the result VARIANTE which is then used to identify the variant description file.

A job which is executed on a group has the following sequence:

Application program	**EDIABAS**	

calls	selects SGBD:	executes job:
apiInit()		
apiJob(GROUP,JOB1,...)		
	GROUP.PRG	INITIALISIERUNG IDENTIFIKATION (of the variant) ENDE
	VARIANTE.PRG	INITIALISIERUNG JOB1

3.5. Filenames

Files that are used by EDIABAS have defined filename extensions which are mandatory and are different for group and variant description files.

File type	Variant	Group
SGBD object file	*.PRG	*.GRP
BEST/2 source file	*.B2V	*.B2G
BEST/2 preprocessor file	P2V	. P2G
BEST/1 source file	*.B1V	*.B1G
BEST info file	*.BIV	*.BIG
BEST/1 map file	*.M1V	*.M1G

4. Tutorial

This tutorial uses a realistic example to show you how to create a BEST/2 description file. We have tried to cover as many aspects of programming as possible. The use of the development tools is also explained at a number of points.

You can work through this tutorial in two ways:

1. You can either use the files supplied by the BEST DEVELOPMENT KIT, or

2. You can edit the files as you go, starting from the DEMO0.B2V file in section 4.2.

With the first method you are prompted what file to load for the section. With the second method all the changes in this manual are listed and shown like this:

```
-> // This line must be added to the previous version.
<- // This line must be removed from the previous version.
<-> // This line must be changed in the previous version.
```

You will find a detailed description of the BEST/2 functions used in this tutorial in [1] which fully describes all BEST/2 library functions with their call parameters and an example.

4.1. Preconditions

For this tutorial a development of ECU description files under MS-WINDOWS is provided. The development environment BestBoard and the Source Code Debugger BestView is only available for MS-WINDOWS.

In order to walk-through the sequences shown in this tutorial, the EDIABAS-RUNTIME-SYSTEM and the BEST-DEVELOPMENT-KIT must be installed.

The following settings has to be done in the EDIABAS configuratioin file EDIABAS.INI:

```
EcuPath = C:\EDIABAS\TUTORIAL
TracePath = C:\EDIABAS\TUTORIAL
SimulationPath = C:\EDIABAS\TUTORIAL
```

Simulation = 1

Assumption is made that EDIABAS has been installed in directory C:\EDIABAS. If this is not the case, the EDIABAS configuration file EDIABAS.INI must be adapted. The configuration setting C:\EDIABAS\TUTORIAL is to be replaced by <EDIABAS-Verzeichnis>\TUTORIAL for the configuration elements EcuPath, TracePath and SimulationPath. Moreover, this must be replaced by the path of the EDIABAS directory everywhere where C:\EDIABAS can be read in this description.

In addition, the following settings are to be kept:

The environment variable PATH should contain C:\EDIABAS\BIN.

The directory C:\EDIABAS\TUTORIAL must not be write-protected.

The tutorial describes how to program ECU communication. In order to enable this independent of the hardware used and the available ECUs, the ECU Simulator contained in EDIABAS is used. Refer to [2] for a detailed functional description concerning this. Files DEMO*.SIM and EDIC.SIM are necessary for this purpose.

The individual development phases are represented by prepared description files in directory C:\EDIABAS\TUTORIAL. A total of 9 steps are involved and the corresponding files DEMO0.B2V to DEMO8.B2V.

In order to continue with the next step, now start MS-WINDOWS.

4.2. Getting started

To begin developing our SGBD, first start the BestBoard routine \EDIABAS\BIN\BESTBRD.EXE for WIN16 or \EDIABAS\BIN\BESTBD32.EXE for WIN32 respectively. Provided your settings are correct you will see the dialogue box "Notify Test Environment" which shows that the control unit simulator is activated in the selected ECU directory. The simulation file directory and the simulated interface are displayed, also whether the traces of API and IFH are active.

Acknowledge this message and you will see the BestBoard main window. All the actions needed to create and test an SGBD can be controlled from this program.

To open the raw version of our SGBD, select the FILE/OPEN menu or press the appropriate key (second from left, see also BestBoard description).

Continue with the next point by selecting the C:\EDIABAS\TUTORIAL\DEMO0.B2V file.

4.3. Step 1: The frame

Opening the file automatically calls the text editor NOTEPAD.EXE. This displays the selected file - in this case DEMO0.B2V (The copyright message contained in this file is not reproduced in this manual). The file contains a frame for an SGBD that is blank but which already has comments:

```
/////////////////////////////////////////////////////////////////////
//                              H E A D E R
/////////////////////////////////////////////////////////////////////
ecu     : ;
origin  : ;
revision: ;
author  : ;
comment : ;

/////////////////////////////////////////////////////////////////////
//                              C O N S T A N T S
/////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////
//                              J O B S
/////////////////////////////////////////////////////////////////////

job(name      : INITIALISIERUNG;
 comment      : Initializing;
 comment      : This job is automatically called by EDIABAS when;
 comment      : first accessing an SGBD. It is not called with;
 comment      : subsequent accessing of the same SGBD. In;
 comment      : INITIALISIERUNG all functions are called which are:
 comment      : only needed once, before communication with a CU.;
 result: DONE;
   type: int;
   defrslt    : ;
   comment    : Values: 0 = initializing failed;
   comment    : Values: 1 = initializing successful;
    )
{
 DONE = 1;
}

/////////////////////////////////////////////////////////////////////

job(name      : ENDE;
 comment      : De-initializing;
 comment      : This job is automatically called by EDIABAS after;
 comment      : the last access to the SGBD. It is used to;
 comment      : de-initialize the interface hardware if required;
```

```
    )
{

}

//////////////////////////////////////////////////////////////////////
```

The file contains a header (that must be filled out) and two jobs. These are the standard jobs INITIALISIERUNG and ENDE and are called automatically by the runtime system when a description file is loaded (INITIALISIERUNG) or unloaded (ENDE).

You will see that a job's header contains its name and at least one comment line. It also lists and comments all the parameters and results of the job. During compiling, these headers are incorporated in the objects and can be read out with the XTRACT tool. The header consists of n entries in the form: "keyword: *data*;". Here is a description of these keywords:

- **name** Name of the job. The name with which the application program addresses the job.
- **comment** Comment line. Contains any desired comment (at least 1 line).
- **result** Beginning of the result block. Must be followed by the keywords **type**, **defrslt** (optional) and **comment**(s). *data* gives the name of the result. The application program and the job can access the result with this name.
- **argument** Beginning of a parameter block. Must be followed by the keywords **type** and **comment**(s). *data* gives the name of the parameter. The application program and the job can access the parameter with this name. Only the parameter sequence is important for the application program.
- **type** Type of parameter or result, the following values of data are allowed:
  - **[unsigned] char** byte or character (8 bit)
  - **[unsigned] int** word or integer (16 bit)
  - **[unsigned] long** double word or long integer (32 bit)

- **string** zero-terminating text string (1024 bytes max.)
- **real** double value
- **data** byte field (1024 bytes max.)
- **defrslt** Optional default value for a result.

In the INITIALISIERUNG job, the result DONE is assigned the value 1. This result **must** be present in this job as it tells the runtime system whether the job was executed successfully or not.

Continue with the next point by selecting the C:\EDIABAS\TUTORIAL\DEMO1.B2V file.

4.4. Step 2: Minimum version

The file now opened can already be compiled. It attempts to communicate with the interface. To do this the following lines must be inserted or changed:

```
/////////////////////////////////////////////////////////////////////
//                                      H E A D E R
/////////////////////////////////////////////////////////////////////
<-> ecu     : EDIABAS BEST/2 Demo control unit;
<-> origin  : softing.SAG.HD;
<-> revision: 1.1;
<-> author  : softing.SAG.HD;
<-> comment : demonstration file for EDIABAS BEST/2;
    ...
 job(name           : INITIALISIERUNG;
       comment      : Initializing;
       comment      : This job is automatically called by EDIABAS when;
       comment      : first accessing an SGBD. It is not called with;
       comment      : subsequent accessing of the same SGBD. In;
       comment      : INITIALISIERUNG all functions are called which;
       comment      : are only needed once, before communication with;
       comment      : a CU.;
       result: DONE;
       type         : int;
       defrslt      : ;
       comment      : Values: 0 = initializing failed;
       comment      : Values: 1 = initializing successful;
    )
    {
->      open_communication();    // make connection to interface

       DONE = 1;
}

/////////////////////////////////////////////////////////////////////

 job(  name         : ENDE;
       comment      : De-initializing;
       comment      : This job is automatically called by EDIABAS;
       comment      : after the last access to the SGBD. It is used to;
       comment      : de-initialize the interface hardware if required;
    )
    {
->      open_communication();    // make connection to interface

    }
```

The header now contains the following information:

- ecu: explicit meaning and function of this description file
- origin: name of first creator

- revision: version code (RevMajor, RevMinor)
- author: author of the last revision (more than 63 characters are ignored)
- comment: any desired comment (n line comment: ...;)

These entries are incorporated in the object when compiled.

Since this SGBD wants to communicate with a control unit (actually with the interface in between) the function **open_communication** must be called in the INITIALISIERUNG and ENDE jobs. This function attempts to open a communication channel to the interface. No special error handling is needed. If an error occurs during this function call, the job routine is aborted at this point and an appropriate error message is sent to the application program. This applies to **all** communication functions.

To compile the created file, start the compiler BEST2WIN in the PROJECT/COMPILE menu. The compiler's icon can be seen on the Windows screen background while it is working. Any errors that occur during compiling are subsequently shown in the application area of BestBoard and the Editor is loaded with the appropriate file. All errors can be shown in sequence using the PROJECT/NEXT ERROR or PROJECT/PREVIOUS ERROR menu (optional also with the 6th and 7th toolbar keys) and corrected in the Editor. An audible signal and a blank application area indicate that the last error has been processed and you may now restart the compiler. When the compiler shows no further error messages a runnable SGBD object file has been generated.

Continue with the next point by opening the C:\EDIABAS\TUTORIAL\DEMO2.B2V file.

4.5. Step 3: Control unit parameters

The filed now opened will also prepare the hardware interface for communication with the control unit. First the constants for the parameters and telegrams are defined:

```
  ////////////////////////////////////////////////////////////////////
//                        C O N S T A N T S
////////////////////////////////////////////////////////////////////
-> //         *** SG - PARAMETERS ***
-> //
-> //         Concept              1
-> //         Baudrate          9600
-> //         Wakeup address    0xaa
-> //         Wakeup time       0 ms
-> //         Idle time         0 ms
-> //         Timeout time      2000 ms
```

```
-> //         Regeneration time    500 ms
-> //         Answer length            -2 (2nd byte in answer telegram)
-> //         Control byte       3 (3rd byte in answer telegram)
-> //         0x80: 3rd byte in send telegram must be combined
-> //         with it to represent a valid third answer byte
-> //
-> int PARAMETER[]  = {1,9600,0xaa,0,0,2000,500};
-> int AWLEN[]            = {-2,0};
-> int CTRLPOS      = 3;
-> int CTRLOR       = 0X80;
->
-> //         *** SG - TELEGRAMS ***
-> //
-> char TEL_ID_READ[]           = { 0xAA,0x00,0x04,0x01 };
-> char TEL_TC_READ[]           = { 0xAA,0x00,0x04,0x02 };
-> char TEL_TC_CLEAR[]          = { 0xAA,0x00,0x04,0x03 };
-> char TEL_RPM_READ[]          = { 0xAA,0x00,0x05,0x04,0x01 };
-> char TEL_TEMP_READ[]         = { 0xAA,0x00,0x05,0x04,0x02 };
-> char TEL_VOLT_READ[]         = { 0xAA,0x00,0x05,0x04,0x03 };
-> char TEL_PARAM_SET[]         = { 0xAA,0x00,0x05,0x05,0xFF };
-> char TEL_END_DIAG[]          = { 0xAA,0x00,0x04,0x0F };
```

The parameters define how the interface will process the telegrams on the diagnostic bus. The individual parameters are explained in the comment before the definition of the parameter fields.

CTRLPOS and CTRLOR are constants which are needed again and again in this SGBD and so they are defined at the beginning of the file to make the file easier to read.

The telegrams are diagnostic jobs defined in the control unit specification.

In the INITIALISIERUNG job the communication parameters are sent to the interface:

```
  ////////////////////////////////////////////////////////////////////
    {
        open_communication();          //make connection to interface
->      set_repeat_counter(2);               // set repeat counter
->      set_communication_pars9PARAMETER);   // set communication
->      set_answer_length(AWLEN);            // parameters
  DONE = 1;
    }
```

The function **set_repeat_counter** ensures that each failed control unit communication is repeated twice. The functions **set_communication_pars** and **set_answer_length** use the predefined constants to load the communication parameters to the interface. These functions and their parameters are described fully in [1].

Continue with the next point by opening the C:\EDIABAS\TUTORIAL\DEMO3.B2V file.

4.6. Step 4: Control unit communication

The file DEMO3.B2V can already communicate with the control unit. To do this properly, two jobs are appended at the end of the SGBD:

```
-> ////////////////////////////////////////////////////////////////////
->
-> job( name         : READ_IDENT;
->      comment      : read out identification data;
->      result       : IDENT_STRING;
->      type         : string;
->      defrslt      : ;
->      comment      : string that identifies the control unit;
->      result       : IDENT_VERSION;
->      type         : string;
->      defrslt      : ;
->      comment      : control unit version number (1.1);
->    )
-> {
->       unsigned char answer[];
->       IDENT_STRING="*****";
->       IDENT_VERSION="*****";
->
->       send_and_receive(answer,TEL_ID_READ);
->
-> }
->
-> ////////////////////////////////////////////////////////////////////
->
-> job( name         : CLEAR_TC;
->      comment      : clear fault memory;
```

```
->      result       : STATUS;
->      type         : string;
->      defrslt      : ;
->      comment      : Text: "Fault memory (NOT) cleared";
->    )
-> {
->      unsigned char answer[];
->      unsigned char statusText[];
->
->      send_and_receive(answer,TEL_TC_CLEAR);
->
-> }
->
-> ////////////////////////////////////////////////////////////////
```

The job READ_IDENT is used to read the identification data, supplying it as two text results, IDENT_STRING and IDENT_VERSION, to the calling application program. The results are pre-assigned the values "****" in the job. the buffer answer[] is defined to receive the control unit answer. A length is not given: all fields are 1024 **bytes** long in BEST/2. Calling the function **send_and_receive** sends the predefined TEL_ID_READ to the control unit and the answer is expected in the answer field.

The job CLEAR_TC6 clears the control unit's fault memory by sending the telegram TEL_TC_CLEAR to the control unit. The answer is expected in the locally defined buffer answer. The field statusText is also defined - this is where the result text for the result STATUS will be generated.

The job ENDE is expanded by sending the diagnostic end telegram TEL_END_DIAG:

```
   {
->      unsigned char answer[];
->
        open_communication();    //make connection to interface
->      send_and_receive(answer,TEL_END_DIAG);
   }
```

Here again, the answer is expected in the answer field.

To test the changes made so far, we will now compile the SGBD. You can ignore the 2 warnings. They just say that the variable statusText and the result STATUS were not used in the CLEAR_TC job. Provided the compile run is OK the description file can be tested with the debugger. To do this, start the BestView debugger with the PROJECT/DEBUG menu or the 5th key from left.

Having started the debugger you are asked to select the job. Select READ_IDENT for the first attempt and press OK to confirm your choice. You will see the SGBD source code in the window. Scroll down and position the cursor on line 109. Use F9 (DEBUG/TOGGLE BREAKPOINT to set a breakpoint in this line, i.e. job processing will be interrupted at that point. Hit F5 (RUN/GO) to start the selected job. A window with a counter shows the number of processed internal working cycles. When the run reaches line 109 it stops and the current line is marked. This line has not been processed yet. You can now continue processing line by line by pressing F8 (RUN/STEP). The debugger allows you to display

variables and result contents during line-by-line processing or when a breakpoint is reached. You will find further details in BestView's online help (HELP/CONTENTS).

To run the job to the end without stopping just hit F5 again (RUN/GO). When the job has finished you should see a window with the results identified by the job. In this case it should contain the following:

```
VARIANTE         = DEMO3.PRG
OBJECT           = DEMO3
SETS             = 1
UBATTCURRENT     = UNKNOWN      UBATTHISTORY     = UNKNOWN
IGNITIONCURRENT = UNKNOWN      IGNITIONHISTORY = UNKNOWN
JOB STATUS       =

SET   1:  RESULT   1:  [ TEXT    ]  IDENT_STRING = "*****"
          RESULT   2:  [ TEXT    ]  IDENT_VERSION = "*****"
```

The operation described above can be repeated as often as you like with this or any other job.

Continue with the next point by opening the C:\EDIABAS\TUTORIAL\DEMO4.B2V file.

4.7. Step 5: String and Data functions

This version evaluates the answer telegrams sent by the control unit. For this purpose the job READ_IDENT is added to the SGBD as follows:

```
   {
->      unsigned char answer[];
->      unsigned char idString[];
->      unsigned char verString[];

 IDENT_STRING="*****";
 IDENT_VERSION="*****";

->      send_and_receive(answer,TEL_ID_READ);

->      // valid answer ?
->      if (answer[CTRLPOS]==TEL_ID_READ[CTRLPOS] | CTRLOR) {
->
->            // Make IDENT_STRING
->            datacopy(idString,answer,4,13);
->            strrevers(idString);
->            IDENT_STRING=idString;
->
->            // Make IDENT_VERSION
->            dataclear(idString);
->            bcd2ascii(verString,answer,17,1);
->            strcopy(idString,verString);
->            strcat(idString,".");
->            bcd2ascii(verString,answer,18,1);
->            strcat(idString,verString);
->            IDENT_VERSION=idString;
->      }
   }
```

To compute the text results the buffers idString and verString are defined. After checking the control byte in the answer telegram, 13 characters are copied from the telegram from the 4th position. The string must also be reversed with the **strrevers** function so that the result is shown correctly.

The control unit version is a 4-digit BCD number in the telegram from address 17. The first BCD number is read in (**bcd2ascii**) and copied to the idString buffer. A full-stop is appended to separate the version number and then the second BCD number is appended.

In the TC_CLEAR job the test result Status should be assigned the string "fault memory cleared" or "fault memory NOT cleared" irrespective of the returned control byte:

```
        unsigned char answer[];
        unsigned char statusText[];

->      statusText="fault memory ";

        send_and_receive(answer,TEL_TC_CLEAR);

->      // valid answer ?
->      if (answer[CTRLPOS]!=TEL_TC_CLEAR[CTRLPOS] | CTRLOR))
->          strcat(statusText,"NOT");
->
->      strcat(statusText, "cleared");
->      STATUS=statusText;
    }
```

If you now compile the file created, starting the debugger as described in the previous section, you will see the following output in the Results window when the READ_IDENT job is finished:

```
VARIANTE         = DEMO4.PRG
OBJECT           = DEMO4
SETS             = 1
UBATTCURRENT     = UNKNOWN     UBATTHISTORY    = UNKNOWN
IGNITIONCURRENT = UNKNOWN     IGNITIONHISTORY = UNKNOWN
JOB STATUS       =

SET   1:  RESULT   1:  [ TEXT    ]  IDENT_STRING  = "ICH BIN SG AA"
          RESULT   2:  [ TEXT    ]  IDENT_VERSION = "18.29"
```

Continue with the next point by opening the C:\EDIABAS\TUTORIAL\DEMO5.B2V file.

4.8. Step 6: Result sets

This version will read out and evaluate the control unit fault memory. For this purpose the job READ_TC is added to the SGBD:

```
-> job( name         : READ_TC;
->      comment      : read out fault memory;
->      result       : TC_NR;
->           type    : int;
->           defrslt: ;
->           comment: fault number;
->     )
-> {
->      unsigned char answer[];
->      int           count;
->
```

```
->      send_and_receive(answer,TEL_TC_READ);
->
->      // valid answer ?
->      if (answer[CTRLPOS]==TEL_TC_READ[CTRLPOS] | CTRLOR) {
->
->         count=256*answer[1]+answer[2]-4; // compute number of faults
->
->         while(count--) {      // for all faults
->              TC_NR=answer[4+count]; // assign fault number to result
->              new_set_of_results(); // start new result set
->            }
->      }
-> }
```

This job returns the result TC_NR which simply displays the stored fault code for all faults reported by the control unit. This is done by storing the same result in a number of result sets. In the loop that runs across all faults the start of a new set is marked after the result value is assigned. All results that are assigned subsequently are stored in this new set, and results in previous sets are not changed.

This job returns the following results when it has been processes in the debugger as described above:

```
VARIANTE         = DEMO5.PRG
OBJECT           = DEMO5
SETS             = 5
UBATTCURRENT     = UNKNOWN     UBATTHISTORY     = UNKNOWN
IGNITIONCURRENT = UNKNOWN     IGNITIONHISTORY = UNKNOWN
JOB STATUS       =

SET   1:  RESULT   1:  [ INTEGER ]  TC_NR = +5
SET   2:  RESULT   1:  [ INTEGER ]  TC_NR = +48
SET   3:  RESULT   1:  [ INTEGER ]  TC_NR = +2
SET   4:  RESULT   1:  [ INTEGER ]  TC_NR = +16
SET   5:  RESULT   1:  [ INTEGER ]  TC_NR = +1
```

Five result sets are created, all containing the same result but with different values.

Continue with the next point by opening the C:\EDIABAS\TUTORIAL\DEMO6.B2V file.

4.9. Step 7: Table processing

As well as the above results in the READ_TC job, file DEMO6.B2V returns not only the fault code but also the corresponding fault text. These fault texts are stored in a table:

```
 char TEL_END_DIAG[]      = { 0xAA,0x00,0x04,0x0f };
->
-> //         *** SG - TABLES ***
-> //
-> //         Table of fault location texts
-> table TCText[2] []=
-> {
-> {"NR",       "TEXT" },
-> {"0x01",     "Rear left speed sensor failed"    },
-> {"0x02",     "Rear right speed sensor failed"   },
-> {"0x03",     "Front right speed sensor failed"  },
-> {"0x04",     "Front left speed sensor failed"   },
-> {"0x05",     "Rear left ABS valve failed" },
-> {"0x06",     "Rear right ABS valve failed"      },
-> {"0x07",     "Front right ABS valve failed"     },
-> {"0x08",     "Front left ABS valve failed"      },
-> {"0x10",     "Internal fault"             },
-> {"0xFF",     "undefined fault"                  }
-> };
```

This tables has 2 columns and as many lines as required. The first column "NR" gives the fault code in hexadecimal format. The second column "TEXT" gives the corresponding fault text for the fault number. The last line is the default telegram that is used when the fault code is not found.

The READ_TC job must be changed as follows:

```
        comment      : Fault no;
->      result       : TC_TEXT;
->           type    : string;
->           defrslt : ;
->           comment : fault text;
->     )
-> {
->      unsigned char answer[];
->      char          tableText[];
        int           count;

  send_and_receive(answer,TEL_TC_READ);

  // valid answer ?
  if (answer[CTRLPOS]==TEL_TC_READ[CTRLPOS] | CTRLOR) {

        count=256*answer[1]+answer[2]-4; // compute number of faults

->      tabset("TCTEXT");    // initialize table processing

        while(count--) {   // for all faults

->            tab_suche_index(Nr",answer[4+count]); // search for fault
                                                    // number
->            tabget(tableText,"Text")); // read corresponding
                                         // fault text
              TC_NR=answer[4+count]; // assign fault number to result
              TC_TEXT=tableText;     // assign fault text to result

              new_set_of_results(); // start new result set
  }
```

The command **tabset** initializes table processing and all subsequent table processing commands affect the "TCText" table. The command **tab_suche_index** changes the second argument it receives into a string with the format "0x##". This string is searched in the "NR" column of the table. The search stops when the value is found. The command **tabget** now reads out the corresponding text on the same line in the "TEXT" column. If the search text is not found it reads the text from the last line.

The results of this job look like this:

```
VARIANTE        = DEMO6.PRG
OBJECT          = DEMO6
SETS            = 5
UBATTCURRENT    = UNKNOWN     UBATTHISTORY    = UNKNOWN
IGNITIONCURRENT = UNKNOWN     IGNITIONHISTORY = UNKNOWN
JOB STATUS      =

SET  1: RESULT  1:    [ INTEGER ]   TC_NR = +5
        RESULT  2:    [ TEXT    ]   TC_TEXT = "Rear left ABS valve failed"
```

```
SET  2: RESULT  1:    [ INTEGER ]   TC_NR = +48
        RESULT  2:    [ TEXT    ]   TC_TEXT = "undefined fault"
SET  3: RESULT  1:    [ INTEGER ]   TC_NR = +2
        RESULT  2:    [ TEXT    ]   TC_TEXT = "Rear right speed sensor failed"
SET  4: RESULT  1:    [ INTEGER ]   TC_NR = +16
        RESULT  2:    [ TEXT    ]   TC_TEXT = "Internal fault"
SET  5: RESULT  1:    [ INTEGER ]   TC_NR = +1
        RESULT  2:    [ TEXT    ]   TC_TEXT = "Rear left speed sensor failed"
```

Each result set contains 2 results, the fault number and the fault text.

Continue with the next point by opening the C:\EDIABAS\TUTORIAL\DEMO7.B2V file.

4.10. Step 8: Real processing and result request

In this version of our SGBD a job is added that reads out three status values from the control unit. The application program can determine which results are to be computed and which are not. This can speed up the job run when only one result is required. the new job is appended to the end of the SGBD:

```
-> ////////////////////////////////////////////////////////////////////
->
-> job( name         : READ_STATES;
->      comment      : read out various statuses;
->      result       : ENGINESPEED;
->          type     : unsigned int;
->          defrslt  : ;
->          comment  : speed in RPM;
->          result   : TEMPERATURE;
->          type     : real;
->          defrslt  : ;
->          comment  : temperature in degrees C;
->          result   : VOLTAGE;
->          type     : real;
->          defrslt  : ;
->          comment  : supply voltage in Volts;
->      )
    {
->      unsigned char answer[];        // buffer for CU answer
->      unsigned int tmp;              // aux variable
->      real a;                        // aux variable
->      real b:                        // aux variable
->
->      if(ENGINESPEED) {    // result requested ?
->
->            send_and_receive(answer,TEL_RPM_READ);
->
->            if (answer[CTRLPOS]==TEL_RPM_READ[CTRLPOS] | CTRLOR) {
->                    tmp=256*answer[4]+answer[5];
```

```
->                      tmp/=4;                         // adc16/4
->                      ENGINESPEED=tmp;
->              }
->      }
->      if(TEMPERATURE) {    // result TEMPERATURE requested ?
->
->              send_and_receive(answer,TEL_TEMP_READ);
->
->              if (answer[CTRLPOS]==TEL_TEMP_READ[CTRLPOS] | CTRLOR) {
->                      tmp=256*answer[4]+answer[5];
->                      itor(a,tmp);
->                      ator(b,"100.0");       // adc16/100-40
->                      realdiv(a,b);
->                      ator(b,"40.0");
->                      realsub(a,b);
->                      TEMPERATURE=a;
->              }
->      }
->      if(VOLTAGE) {        // result VOLTAGE requested ?
->
->              send_and_receive(answer,TEL_VOLT_READ);
->
->              if (answer[CTRLPOS]==TEL_VOLT_READ[CTRLPOS] | CTRLOR) {
->                      tmp=256*answer[4]+answer[5];
->                      itor(a,tmp);
->                      ator(b,"1.238"); // 1.238*adc16100/4000
->                      realmul(a,b);
->                      ator(b,"4000.0");
->                      realdiv(a,b);
->                      VOLTAGE=a;
->              }
->        }
-> }
```

The construct if(ENGINESPEED) does not read the value of the result ENGINESPEED, it decides whether this result was requested by the application program (the way in which the application program requests these results is described in [6]). ENGINESPEED returns TRUE if the result must be computed, otherwise FALSE. You can also test this in BestView: the RUN/JOB dialogue contains a field in which the requested results can be entered. If nothing is entered, then all the results are returned. To request the VOLTAGE and ENGINESPEED results enter "Results you want to see:" in the field.

voltage;enginespeed

The instructions in the block if(TEMPERATURE) { ... } are not processed. If nothing is entered then all results are reckoned to have been requested.

If nothing is entered, all results are returned.

When computing the temperature (TEMPERATURE) and supply voltage (VOLTAGE) an integer is read out of the answer telegram (tmp) and changed to a real number by the **itor** function. This real number is then handled by functions because BEST/2 has not operators to process real numbers with.

Continue with the next point by opening the C:\EDIABAS\TUTORIAL\DEMO8.B2V file.

4.11. Step 8: Job parameter

In this final version a job is added to the description file that sends a parameter defined by the application program to the control unit. This parameter is a byte that is inserted into the telegram to the control unit. When the parameter is sent its value is overwritten on the 5th byte in the telegram to the control unit. If no parameter is set the predefined telegram is used:

```
`->
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
////////
->
-> job(name         : SET_PARAM;
->      comment      : set a parameter in the control unit;
->      argument     : PARAMETER;
->           type    : int;
->           comment: value of parameter to be set;
->       result      : STATUS;
->           type    : string;
->           defrslt: ;
->           comment: Text: "Parameter (NOT) set";
->      )
-> {
->      unsigned char answer[];              // buffer for CU answer
->      unsigned char request[];       // buffer for CU_Request
->      unsigned char statusText[];          // buffer for resulttext
->
->      request=TEL_PARAM_SET;         // generate request
->
->      if(exist(PARAMETER)) {         // if parameter exists
->          request[4]=PARAMETER;      // change request
->      }
->      send_and_receive(answer,request);
->
->      statusText="Parameter";        // pre-assign statusText
->      // valid answer ?
->      if (answer[CTRLPOS]!=TEL_PARAM_SET[CTRLPOS] | CTRLOR))
->          strcat(statusText,"NOT");
->
->      strcat(statusText, "set");
->      STATUS=statusText;
-> }
```

The **exist** operator checks whether the parameter PARAMETER has been supplied to the job. the application program must send the parameters in the sequence as defined in the job header. There is no check. The application program supplies the parameters as a cohesive string, with parameters separated by a ;. A parameter ;; is reckoned not to be sent. In BestView you can also send

arguments to the job. To do this you edit the field "Parameters to supplied to job:" (sic) field in the RUN/JOB dialogue.

Examples:

```
;;TEST
```

Only supplies TEST as the third argument to a job.

```
0x0005;0x0001
```

Sends two parameters to a job.

The parameters must be provided with data types, then conversion is automatic. If the first parameter in the above example is a **string** type and the second an **int** type, then the first is seen as a field and the second as a number.

4.12. Binary parameters and binary results

A job can also receive and return binary parameters and results.

Only **one** binary parameter of the **data** type can be defined. This contains a binary field that contains either the complete parameter string or a special data field. API has a special functions (**apiJobData/apiJobExt**, see [6]) for supplying a binary data field.

More than one result of the **data** type can be defined on the other hand, and these can be assigned array variables in binary format, e.g. telegrams.

Example:

```
job(  name         : READ_2_TELEGRAMS;
      comment      : set a parameter in the control unit;
      argument     : SEND_TEL;
        type       : data;
        comment    : telegram to be sent;
      result: RECV_TEL_1;
        type       : data;
        defrslt    : ;
        comment    : 1st receive telegram
      result: RECV_TEL_2;
        type       : data;
        defrslt    : ;
        comment    : 2nd receive telegram
      )
  {
```

```
unsigned char answer1[];       // buffer for CU answer
unsigned char answer2[];       // buffer for CU answer
unsigned char request[];       // buffer for CU request

request=SEND_TEL;              // generate request

send_and_receive(answer1,request);   // CU communication
send_and_receive(answer2,request);

RECV_TEL_1=answer1;            // results
RECV_TEL_2=answer2;

  }
```

5. BEST Tools

This section describes tools for creating description files.

5.1. BEST2WIN

BEST2WIN is used to compile ECU description files written in BEST/2 into an object which can be interpreted by EDIABAS.

BEST2WIN is a Windows program called BEST2WIN.EXE and is consequently started from the program or file manager. We recommend, however, that BEST2WIN be included in a program group.

An on-line help is available in menu "Help" under option "Contents" which extensively explains all operating procedures as well as the BEST2WIN call parameters.

BEST2WIN can also be assigned parameters via the command line:

BEST2WIN [-**C** commandfile] [**-B**] [**-A**] [**-I**] [-**S**] [**-L** *libfile*] [-**Z** *includedir*]

[**-R** *RevMAJ.RevMIN username*] [-**O** errorlogfile]

[**-P** *Paßwortlabel*|*@Paßwortlabelliste*] *sourcefile* [*outdir*]

Compiler option **-L** enables specification of an alternative BEST/2 runtime library. If this option is not used, BEST/2 uses the default file B2RUNTIM.LIB. Search is made in the directory where BEST2.EXE resides.

BEST2WIN supports the use of Include files. These files are first searched in the directory of the BEST/2 source file. With the option **–Z,** the search can be extended to another directory. If several Include directories are indicated, they are searched from left to right, where each directory is labelled with the option **–Z** (-Z incldir1 –Z incldir2 ... –Z incldirN). This option is disregarded if the BEST/2 source file already contained information on the path of the Include file.

The option **_-S** leads to the additional output of the BEST/2 preprocessor file with the ending **.p2v** for variant description files or **.p2g** for group description files, respectively. The BEST/2 preprocessor file contains the entire BEST/2 source code including the content of the BEST/2 Include files.

If compiler option **-A** or **-I** are also specified, either an assembler output (*.b1) or an Infofile with the extension .bix for variant description files (or **.big** for group

descriptions files (or both)) are output in additional to the object file. All mandatory comments from the BES/2 source file are listed in the Infofile.

Compiler option **-R** and its parameters enables a certain revision number and an author to be stored in the object file, overwriting the specifications in the BEST/2 header.

The option **–O** stores an error log file. This file has the extension **.out**. If compiler option **-O** is not specified, an error log file with the name best2err.out is then automatically created in case of an error.

Compiler option **-P** and its parameters enables certain passwords to be stored in the object file. Either a single label to a password can be specified in a file, or an entire list of label. The parameter is characterized by a prefix "@" as file name. Only password labels can be used which were previously made known to the system.

The parameter *sourcefile* describes one or more BEST/2 source files. The optional parameter *outdir* determines the directory where BEST2 is to store the output files.

BEST2WIN command line parameters can be stored in separate text files and made known by the command line parameter **-C** <command line parameter file>.

Example:

```
BEST2WIN demo.b2v

BEST2WIN -R 1.0 Otto *.b2g

BEST2WIN -A -I -S -Z c:\test\ecu\include test.b2v c:\test\ecu

BEST2WIN -L c:\test\ecu\mylib.lib -I test.b2v c:\test\ecu

BEST2WIN -P GEHEIM test.b2v c:\test\ecu
```

The minimum version required by EDIABAS is output for each source files which compiled without errors:

<Quelldatei>: 0 error(s) [EDIABAS <Mindestversion>]

5.2. BestBoard

5.2.1. General Information

BestBoard combines the creation of description files in a single program and controls the Editor, the Compiler and the Debugger. It also lets you run your own application programs for test purposes.

BestBoard is available for WIN32 (BESTBD32.EXE) and WIN16 (BESTBRD.EXE). BestBoard is started from the Explorer or the File Manager respectively. However we recommend that you put BestBoard in a program group.

After calling BestBoard, you must first enter 2 directories:

- ECU Directory:
  Menu: Options ECU directory
  The SGBDs in object format are in this directory, or they should be stored there.
  When the program is started, the EDIABAS configuration element EcuPath is considered and, if available, assumed.

- EDIABAS Directory:
  Menu: Options EDIABAS directory
  This directory contains the installed EDIABAS version with the sub-directories BIN, ECU, API etc.
  When the program is started, the current EDIABAS directory is assumed.

The Editor to be used should also be entered:

The name of the editor must be entered in the Tools/Editor menu with the exact command line. A working directory of the Editor can also be entered optionally. The "Notepad" editor that is generally available under Windows is used as a default.

If the control unit simulator or an EDIABAS trace is activated while starting or stopping BestBoard, you will see a warning together with the set parameters.

5.2.2. Menu overview

File

New	Creates a new description file (BEST/2 source file)
Open	Opens a description file (as source file or object file) The File menu also lists the last 4 description files, the desired description file is opened by selecting a name
Exit	Quit program

Project

Edit	Calls the Editor with the BEST source file
Compile	Compiles the BEST source file by calling BEST2WIN
Debug	Calls the BEST debugger BestView with the BEST source file and BEST object file
Next Error	Displays the next compiler error
Previous Error	Displays the previous compiler error
Run Application	Calls the active application program (see also the section on application programs)

Tools

Applications	Menu for application program management: activates, clears, adds, changes and calls application programs
Editor	Identifies the Editor to be used
API-Trace	Turns API Trace On/Off
IFH-Trace	Turns IFH Trace On/Off
ECU Simulation	Turns IFH control unit simulator On/Off

Options

ECU Directory	Identifies the ECU directory N.B.: The environment variable ECU is not changed!
EDIABAS-Dir.	Identifies the EDIABAS directory N.B.: The environment variable EDIABAS is not changed!
Auto Editor Act.	Automatic activation of the editor when opening the BEST source file.
Save Configur	Stores the current BestBoard configuration when you quit BestBoard (file BESTBD.INI or BESTBRD.INI in the Windows directory)

Window

Default	Sets the default position and size of the BestBoard main window

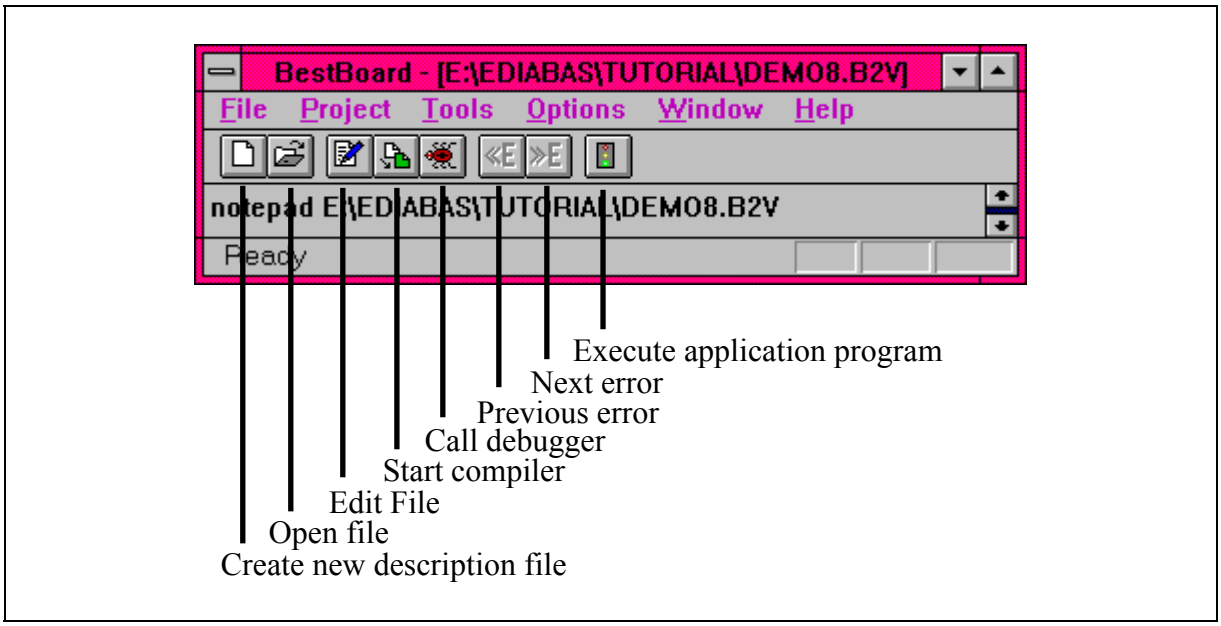
Always On Top	Turns on/off the option of always positioning BestBoard in the desktop foreground
Toolbar	Turns Toolbar On/Off
Status Bar	Turns status display On/Off

Help

About	Displays BestBoard version

5.2.3. Toolbar

The Toolbar is intended to simplify the selection of the main menu options. Hitting a button in the Toolbar selects only the corresponding menu.



"Pressed" buttons indicate that an action has started (e.g. Compiler running).

5.2.4. Application programs

In BestBoard it is possible to integrate any desired programs that you need to develop or test description files. BestBoard can manage several application programs simultaneously. BestBoard always starts the active application program. The active application program is the application program selected in the TOOLS/APPLICATIONS menu.

The name and the command line must be entered for each application program. If you wish you can also specify the working directory in which the application program is started.

Various replacements can be entered on the command line (e.g. current object file, current ECU directory etc.).

In BESTBD32.EXE you can specify WIN32- and WIN16- programs.. In BESTBRD.EXE you can specify only WIN16 programs.

N.B.:

Application programs can only be called when a description file is opened/generated in BestBoard.

5.2.5. Developing a description file with BestBoard

1. Creating BEST source file:

- If source file does not yet exist: File/New menu or Toolbar button
- If source file already exists: File/Open menu or Toolbar button
- Enter source file in Editor (is started automatically)

2. Compiling BEST source file:

- Project/Compile menu or Toolbar button
- If compile errors occur:
  Correct source file in the Editor (is started automatically), the relevant line number and the error message are output in the BestBoard working area. You can switch to the other error messages with the Project/Next Error and Project/Previous Error menus. An audible warning tone and an empty working area signal the end of the error list.
- After correcting the description file in the Editor: Repeat 2.

3. Testing description file with Debugger

- Project/Debug menu or Toolbar button
- With logic errors:
  Start Editor with Project/Edit or Toolbar button and edit source file, then repeat 2.

4. Testing description file with one or more application programs

- Select application program from Tools/Applications menu

- Start application program with Project/Run Application menu or Toolbar button
- With logic errors proceed as in 3.

5.3. BestView

5.5.1. GENERAL INFORMATION

BestView is a Source Level Debugger for testing SGBD's written in BEST/2. These files must exist both as source code and as object files. Alternatively, object or BEST/1 files can be tested but without the source code debugging facility.

BestView is available for WIN32 (BESTVW32.EXE) and WIN16 (BESTVIEW.EXE). Both versions are started either from BestBoard or from the Explorer or file manager.

With BestView you can run jobs line by line and view variables and analyze the job results during the run.

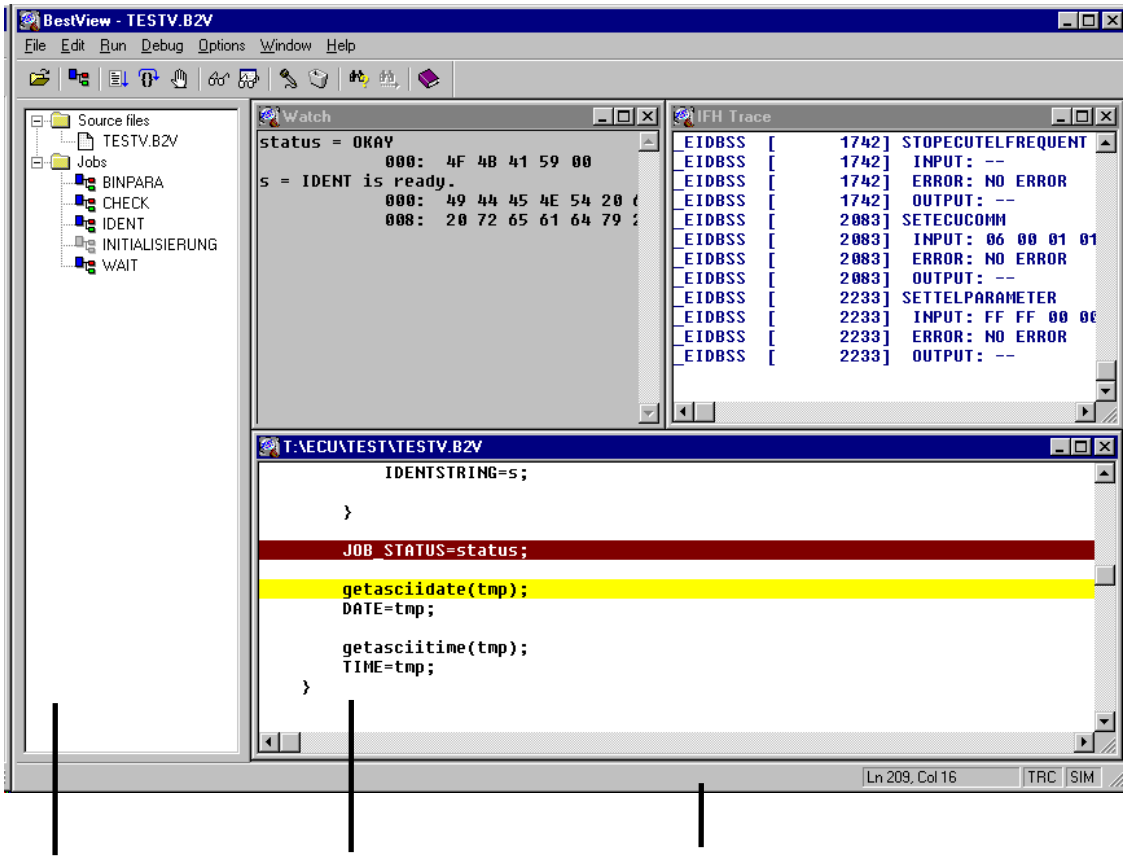
You can get detailed explanations of all the user operations and the BestView call parameters by selecting the "Contents" option from the "Help" menu.

In the following chapters, the WIN32 version is described in detail.

5.5.2. User Interface

BestView for WIN32 has a new user interface different from the WIN16 version with new and altered workspaces:

5.5.2.1. Toolbar



Pro Sou Statusz

BestView now has a toolbar with buttons for the following functions:

- choosing SGBD
- choosing job

- starting/continuing job
- continuing job by 1 instruction
- setting or deleting breakpoint
- opening variable, parameter, or result dialog
- opening watch variable dialog
- displaying IFH trace file
- deleting IFH trace file
- opening text search dialog
- repeating text search
- online help

The toolbar is implemented as a docking window and can also be positioned outside the BestView workspace.

5.5.2.2. Status Bar

In the status bar, BestView displays additional information. On the right side, an active simulation and/or IFH trace is indicated.

If the source, IFH and result windows are active, the current position within the window (row, column) is also displayed in the status bar.

5.3.0.1. Project Window

The Project Window provides information on the content of the currently loaded description file. The folder "Source Files" lists all source files involved in the compilation process (B2V file + Include files). The "Jobs" folder contains all jobs available in the description file (except virtual jobs and jobs in basic files).

By clicking once on an entry in the Project Window, it is possible to display the corresponding source file or the first instruction of the corresponding job in the Source Window, respectively.

A double-click on a job within the Project Window will open the job selection dialog, unless this job is one of the jobs last started (see chapter Menu Run-Job). In this case, a double-click will start the job immediately.

The Project Window can be used as a miniature window or as a docking window (see fig.). As a miniature window, it can also be positioned outside the BestView workspace without changing it. As a docking window, it can dock with the left, right, top or bottom part of the BestView workspace; the size of the BestView workspace decreases accordingly. It is possible to switch between miniature and docking window representation by double-clicking on the frame of the Project Window. A new docking position can be selected by simply drawing the Project Window to the desired position on the BestView frame.

The width and height of a "docked" Project Window are determined by the size of the Project Window in miniature window representation. For changing the size of the "docked" Project Window, you have to change to the miniature window display, alter the window size and then change back to the docking display.

5.5.2.4. Source Code Window

In the Source Code Window, the source code of the SGBD is displayed. Here, breakpoints (with brown background) can be set. The line to be executed during the next program step has a yellow background.

5.5.2.5. Watch Window

In the Watch Window, the content of variables is displayed.

5.5.2.6. Results Window

In the Results Window, the content of the job results is displayed.

5.5.2.7. IFH Trace Window

In the IFH Trace Window, the content of the IFH trace file is displayed. After each job execution and each job interruption (e.g. after Run-Step), the Trace Window is automatically updated.

5.5.3. Selecting a Description File

Description files can be selected with the ”File Open“ menu. BEST/2 source files (with the ending .B2V or .B2G) can be opened as well as BEST/1 source files (with the ending .B1V or .B1G) and executable object files (with the ending .PRG or .GRP). A description file can also be opened by drag&drop; the file must be dragged from the Explorer to the workspace of BestView.

Debugging is possible only if a BEST/2 source file is opened. With a BEST/1 source file or an object file, the jobs can only be executed.

If a source file is selected, BestView searches the corresponding object file in the directory selected with ”ECU-Directory“. If you do not want this, you can set the ”ECU-Directory“ option to '.' In this case, BestView searches the object file in the directory of the source file. Changing the ECU directory is only possible if no description file is loaded in BestView.

If an object file is selected, BestView behaves like the normal EDIABAS runtime system. If a group is selected, the job IDENTIFIKATION in the group file is executed automatically; then the selected job in the variant file determined in this manner is executed. In case of source file selection, a group description file behaves in the same way as a variant description file, i.e. the IDENTIFIKATION job has no particular significance and can be examined at any time.

When a description file is opened, it is displayed in a Source Window; if, on the other hand, an object file is selected, only a minimized Object Window is displayed. A description file can be based on several BEST/2 source files (modules). With the ”File View“ menu, it is possible to switch between the different modules.

BestView permits automatic opening of a Project Window displaying all jobs and BEST/2 source files (modules) of the loaded description file. The Project Window can be used as a miniature or a docking window. With the miniature display, the Project Window can be positioned in any part of the window

desktop without influencing the workspace of BestView. With the docking window display, the workspace of BestView is automatically reduced in size depending on the docking position (left, right, top or bottom). By double-clicking on the Project Window frame, it is possible to switch between the miniature and the docking display. With the ”Window Project“ menu, the Project Window display can be switched on or off.

The linkage of the debugger with a description file is maintained until the file is closed with the ”File Close“ menu, the Description File Window is closed or a new description file is selected.

If a description file consists of several modules (the B2V file opened with File Open + Include files), the same search algorithm is used for the Include file search as with the BEST/2 Compiler:

- For #include instructions with absolute or relative paths, BestView only searches the indicated paths for the relevant Include file. In case of relative paths, the path always refers to the directory of the B2V file opened with “File Open“ (source directory).
- In case of #include instructions without indication of path, BestView first searches the directory of the B2V file opened with ”File Open“ for the corresponding Include file. With the ”Options Include-Path“ menu, the search can be extended to other directories.

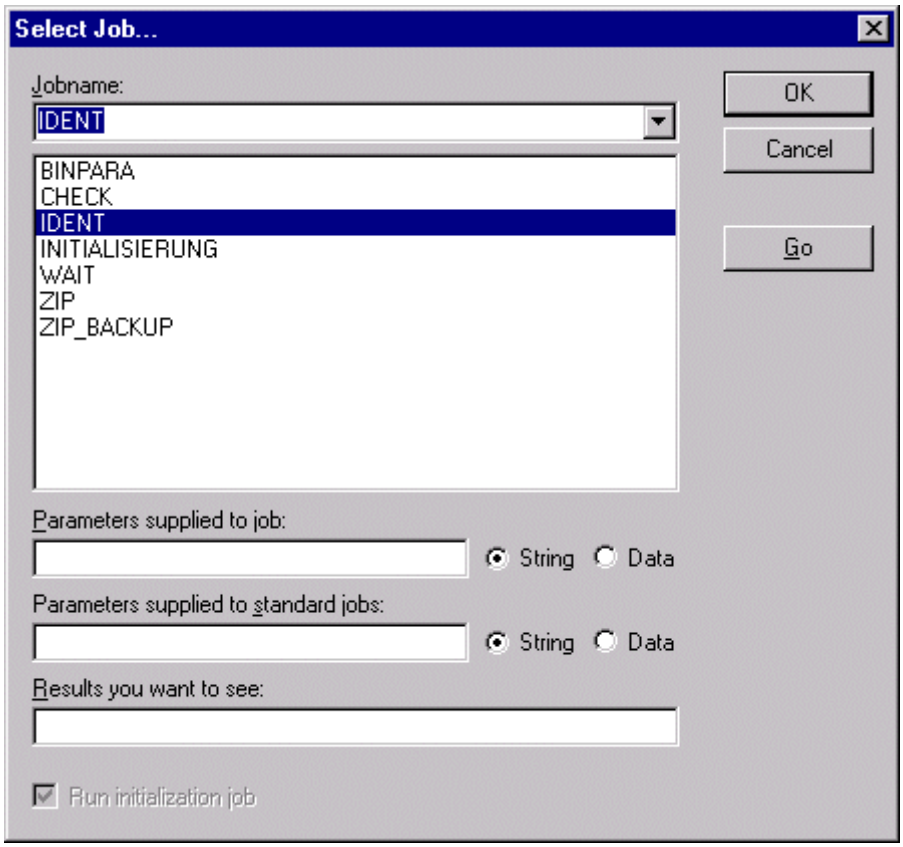
5.5.4. Updating a Description File

Changing a description file without leaving the debugger makes it necessary to update the description file.

This is done easily by again selecting the loaded description file with the “File Open“ menu.

Choose the file already indicated by default.

5.5.5. Selecting a Job



Selection of a job is performed by the ”Run Job“ menu. All jobs of the selected description file are displayed. The job INITIALISIERUNG can be selected as well. In group description files, selection of the job IDENTIFIKATION is also possible.

If a group description file is opened as object file, the entered job name refers to the variant file and not to the group file.

The job to be executed can be configured by the following entries:

- **Job Parameters**

  The data entered here will later be passed to the job to be executed. With the ”String“ setting, the job parameters are passed to the job unchanged. With

the ”Data“ setting, the job parameters are converted into a binary data stream before job execution and then passed to the job.

Binary data entry requires the following syntax:

hex,hex,hex,...

As an alternative to the direct entry of binary data, the job can also be provided with binary data specified in a separate text file. If @<text file name> is indicated, the text file will be read in during job execution, converted into a binary data stream and then be passed to the job. For this purpose, the text file must have the following format:

hex,hex,hex,hex,hex,hex,hex,hex

hex,hex,...

- **Standard Job Parameters**

  The data entered here will later be passed to the standard jobs automatically processed by BestView.

  The possibilities of entry are identical with those of the other job parameters.

- **Job Results**

  Here, the results to be generated by the job can be selected. This functionality must be supported by the description file.

  If nothing is entered in this input field, all results are generated.

- **Initialization of the Description File**

  This entry makes it possible to automatically execute the INITIALISIERUNG job directly before execution of the selected job. After the loading of a description file, occurrence of an error or in case of job abort, the INITIALISIERUNG job must always be carried through.

BestView saves the last 8 job names including all job configurations. By means of the drop-down list, a job carried out previously can be selected with the corresponding job configurations.

With ”Go“, the job can be started directly from this dialog box.

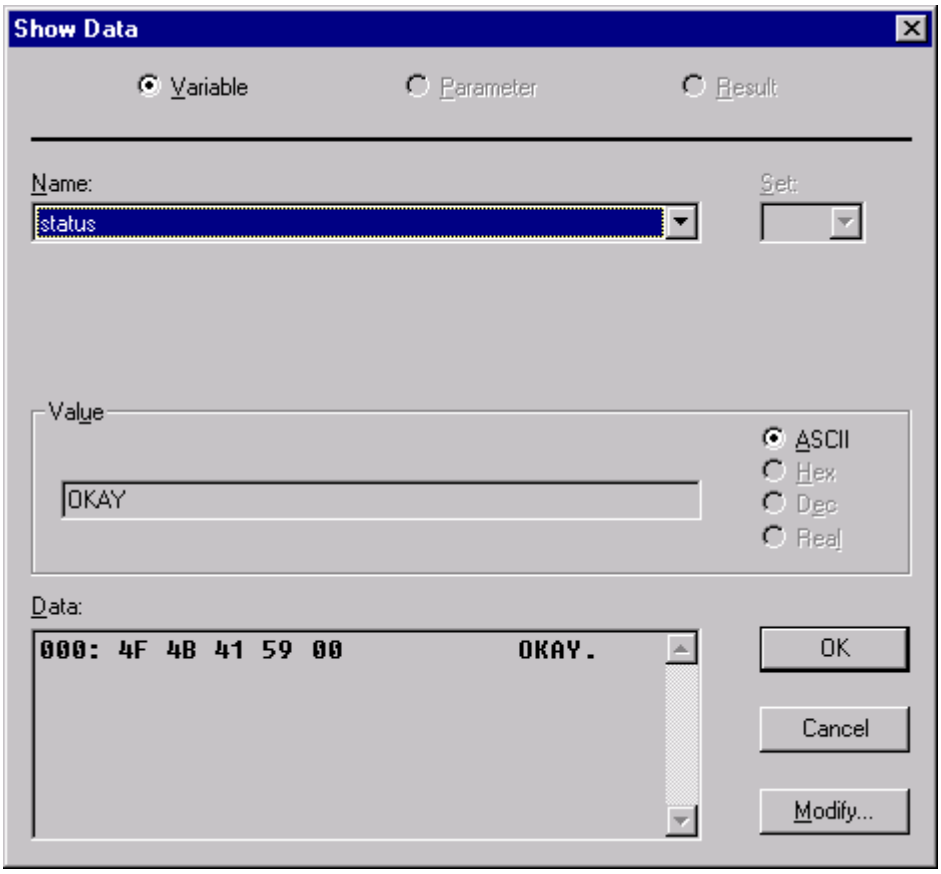
5.5.6. Setting Breakpoints

Breakpoints are set in the Source Window. For this purpose, the cursor is placed on the line in the Source Window in which the breakpoint is supposed to appear.

With the F9 key, the breakpoint in this line is activated. The line color changes. By pressing F9 again, the breakpoint is deactivated.

Breakpoints can also be set and deleted with the mouse. To do this, place the mouse pointer on the desired line. A double-click with the left mouse button will activate or deactivate the breakpoint in this line, respectively.

With the menu item ”Debug Clear Breakpoints“, all breakpoints are deleted.



5.5.7. Display and Alteration of Variables

Variables, results and parameters can be displayed with the menu item ”Debug Show Data“.

A dialog box appears in which the kind of data to be displayed can be selected. For the results, the desired set must be selected in addition to the name.

All data are displayed as values and as binary data. The display format of the value can be adapted according to the requirements.

If the displayed value is a variable, it can be changed from the dialog box by a click on the Modify button or by means of the menu item ”Debug Modify Variable“ in a separate dialog box.

During debugging, a variable can be continuously monitored in a separate window. For this purpose, it must be included in the watch list with the menu item ”Debug Watch Variable“. It can also be removed from there.

If the mouse points to a variable or a parameter in the Source Window, the variable and parameter are automatically displayed. Data variables are shown in hexadecimal form; the display is limited to max. 32 bytes.

5.5.8. Execution of a Job

Before the execution of a job, the description file is initialized if this was defined during job selection. For this purpose, the EDIABAS runtime system is initialized and the INITIALISIERUNG job runs.

There are three ways of job processing:

- **Go**

The job runs until either a breakpoint or the end of the job are reached.

- **Step**

The next line in the job is executed.

- **Continue to Cursor**

The job is executed down to the line where the cursor is located. If a breakpoint is set before the highlighted line, the job is only executed to this breakpoint. It runs until either a breakpoint or the end of the job are reached.

In addition, it is possible to start a job by double-clicking on the corresponding job name in the Project Window. In case of an unknown job configuration, the job selection dialog is displayed instead.

The line being currently processed is highlighted in color.

5.5.9. Menus

5.5.9.1. Menu File

Open

Opens a description file. Both source files and object files can be opened. Debugging is possible only if a BEST/2 source file is opened. With a BEST/1 source file or an object file, the jobs can only be executed.

If a source file is selected, the object file is searched according to the configured ECU directory. If this directory is supposed to be always identical with the source file directory, the directory in the Options menu must be set to '.' If ”Use EDIABAS Configuration“ is selected, the ECU directory from the EDIABAS configuration file (EDIABAS.INI) is used.

Close

Closing the loaded description file. The linkage of the debugger with the loaded description file is cleared.

As long as a description file is loaded, it may not be changed in an Editor. Opening a new description file and leaving BestView will automatically close the currently loaded description file.

View

Selection of the module to be displayed. If a description file consists of several BEST/2 source files (modules), another module of the description file can be displayed in this manner.

The module to be displayed can also be selected via the Project Window.

Exit

Leaving BestView.

5.5.9.2. Edit Menu

Copy

Copying a text selected in the active window to the clipboard.

Select All

Selecting the entire text of the active window.

Find

Searching a text in the active window. If the option ”Match Uppercase/Lowercase“ is activated, the case of character of the searched text must be matched precisely.

The text search starts at the current cursor position.

Find Next

Text search is continued at the current cursor position.

Goto Line

Goes to a specific text line. By entering the desired line number, it is possible to go to the corresponding text line in the source file.

5.5.9.3. Run Menu

Job

Selection of the job to be executed. All jobs within the description file can be selected. In addition, the job parameters and desired results can be indicated. With the ”Go“ button, the jobs can be started immediately.

Go

Start or restart of job processing. The job is executed until either a breakpoint or the end of the job are reached.

Continue to Cursor

Start or restart of job processing. The job is executed down to the line where the cursor is located. If a breakpoint is set before the highlighted line, the job is only executed to this breakpoint. It runs until either a breakpoint or the end of the job are reached.

Step

Start or restart of job processing. The next code line is executed.

Restart

Restarts the selected job if it has not been finished yet.

5.5.9.4. Debug Menu

Toggle Breakpoint

Setting or deleting a breakpoint. If no breakpoint is set in the current line, it is set now; otherwise, it is deleted.

The same can be achieved by double-clicking with the left mouse button on the corresponding line.

If it is not possible to set a breakpoint, an acoustic signal is given.

Clear Breakpoints

Deletes all breakpoints.

Show Data

Display of a variable, a parameter or a result. A dialog box is displayed where the values of all currently valid variables, parameters or results and their binary representation can be displayed.

In this dialog box, the variable values can also be changed.

Modify Variable

Display and modification of a variable. All variables which are currently valid can be displayed and changed. A dialog box appears which contains the variable value and its binary representation.

The value of single variables (no arrays) can be displayed in decimal, hexadecimal or ASCII representation.

The variable value can be changed in all three kinds of representation. By pressing the Update key, the binary display is updated.

The variable can also be changed in binary representation; the display format, however, must be observed.

After ”OK“ or ”Return“, the altered variable value is stored.

After ”Cancel“ or ”ESC“, the process is terminated without variable alteration.

Watch Variable

Monitoring of a variable. The values of selected variables are displayed in a separate Watch window and updated continuously, i.e. after each step.

5.5.9.5. Options Menu

ECU Directory

Here, the directory can be set where the debugger searches the object files. If the debugger is to search the Source Directory, ”.“ must be entered.

If the option ”Use EDIABAS Configuration“ is activated, the ECU Directory selected via the EDIABAS configuration file EDIABAS.INI is used.

Include Path

Here, additional directories can be indicated where BestView searches Include files. Directories must be separated by ”:“. The search always begins in the directory of the BEST/2 source file opened with “File Open”. Subsequently, the Include path is searched from left to right.

For #include instructions with absolute or relative paths, the Include path is not taken into account.

Device

Here, the communication device for EDIABAS can be set.

ECU Simulation

Here, the ECU simulation can be switched on or off; the EDIABAS configuration file EDIABAS.INI remains unchanged. In the status bar, an active ECU simulation is indicated.

IFH Trace

Here, the IFH trace level can be entered and the IFH trace switched on or off, accordingly. The EDIABAS configuration file EDIABAS.INI remains unchanged. In the status bar, an active IFH trace is indicated.

The IFH trace can be displayed via the menu ”Window IFH Trace“.

With the ”Delete File“ button, the IFH trace file can be deleted.

Auto Job Selection

If this option is active, the “Run Job” menu will automatically be executed after opening of a description file.

5.5.9.6. Window Menu

Project

Opens or closes Project Window.

Watch

Opens or closes Watch Window.

IFH Trace

Opens or closes IFH Trace Window.

Cascade

Arrangement of the windows in cascade form.

Tile Horizontally

Symmetrical horizontal arrangement of all windows with the exception of minimized windows.

Tile Vertically

Symmetrical vertical arrangement of all windows with the exception of minimized windows.

Arrange Icons

Uniform display of all minimized windows on the lower left margin.

5.5.9.7. Help Menu

Contents

Calls the help text with its contents.

Index

Calls the help text with its search dialog.

About

Display of the version information for BestView and EDIABAS.

A. Limits and restrictions

ECU description files are subject to the following technical limitations and restrictions:

- Maximum line length of a BEST/2 source file = 5199 characters
- Maximum length of a BEST/2 source file (without extension) = 8 characters
- Maximum length of a basic file name (without extension) = 8 characters
- Maximum length of the author name = 63 characters
  (additional characters are ignored)
- Maximum length of a BEST/2 names = 32 characters
  (additional characters are ignored)
- Maximum length of a job name in the job header = 63 characters
- Maximum length of a argument name in the job header = 63 characters
- Maximum length of a result name in the job header = 63 characters
- Maximum lenght of a comment in the job header = 999 characters
- Maximum number of result records for a job = 65535
- Maximum number of results in a result record = 65535
- Maximum number of string and array variables without a job = 6
- Maximum size of a char arrays = 1023 characters
- Maximum size of an (unsigned) int array = 1023 bytes (511 values)
- Maximum size of a (unsigned) long array = 1023 bytes (255 values)
- Maximum length of a string result = 1023 characters
- Maximum size of a data result = 1023 bytes
- Maximum size of the global data memory = 1023 characters
- Maximum length of a global data memory ID = 32 characters
- Maximum number of passwords = 10
- Maximum length of a password label = 10 characters

- Maximum length of a password = 10 characters
- Maximum number of nested Include files = 8
- Maximum length of a result name = 255

B. REFERENCES

[1] EDIABAS: BEST/2 Function Primer

[2] EDIABAS: Control unit simulator

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

[4] EDIABAS: User Manual

[5] EDIABAS: BEST/2 - Language Description

[6] EDIABAS: API User Manual