



Manual

Decoder Debugger

Product Version v24.1

December 18, 2023

Imprint

PROCITEC GmbH
Rastatter Straße 41
D-75179 Pforzheim
Germany

Phone: +49 7231 15561 0
Fax: +49 7231 15561 11
Email: service@procitec.com
Web: www.procitec.com

Authorised Representative:
Dipl.-Ing. (FH) Dipl.-Inf. (FH) Jens Heyen

Registration Court:
HRB 504702 Amtsgericht Mannheim
Tax ID:
DE 203 881 534

Document ID:
PROCITEC-IMA-DecoderDebugger_E-7f142bd669

All product names mentioned in this text are trademarks or registered trademarks of the respective title-holders.

© 2023 PROCITEC GmbH

All content, texts, graphics and images are copy-righted by PROCITEC GmbH, if not stated otherwise. Reproduction in any form, the rights of translation, processing, duplication, modification, use and distribution by use of electronic systems in whole or part are strictly prohibited.

Subject to technical modifications.

Contents

1. General	1
1.1. Welcome to the Decoder Debugger	1
1.2. Software Environment	1
2. Decoder Adaptation and Development	2
2.0.1. Create Decoder	2
2.0.2. Edit Decoder	3
2.0.2.1. Load Editor	3
2.0.2.2. Menu Bar	5
2.0.2.3. Toolbar	8
2.0.2.4. Shortcuts for Decoder Creation	8
2.0.2.5. Context-Sensitive Help	9
2.0.2.6. Automatic Command Completion	9
2.0.2.7. Automatic Indentation	10
2.0.2.8. Show Parameter Information	10
2.0.2.9. Compile Directory	10
2.0.3. Decoder Source Code Structure	10
2.0.4. Compile and Operate New Decoders	15
2.0.4.1. Start Compiler	15
3. Offline Mode	17
3.1. Offline Operating Concept	17
3.2. Decoder Debugger - Offline Main Window	18
3.3. Producing a Demodulator Output Record	19
3.4. Loading Record File and Decoder	19
4. Online Mode	20
4.1. Operating Concept of Online Debugging	20
4.2. Selecting the Decoder Debugger – Online User Interface	20
4.3. Selecting a Decoder for Debugging	22
5. Running and Analyzing Decoder Programs	24
5.1. Free Run	24
5.2. Breakpoints	24
5.3. Stepping Modes	24
5.3.1. Single Line	25
5.3.2. Single Step	25
5.3.3. Single Package	25
5.4. Restarting the Decoder	25

5.5. Clear Result	25
6. Monitoring Variables	26
6.1. Variables List	26
6.1.1. Variable Types	26
6.1.2. Selecting Variables	27
6.1.3. Changing Variable Values	27
6.2. Watch List	27
6.3. Long Variable Display	28
7. Displays	30
7.1. Buffers	30
7.1.1. Input Package	30
7.1.1.1. Phase	31
7.1.1.2. Package Size	31
7.1.1.3. No. of Channels	31
7.1.1.4. Symbol Size	31
7.1.1.5. Symbol Rate	31
7.1.1.6. Time/Date	32
7.1.2. Input Buffer	32
7.1.3. Output Buffer	32
7.2. Demodulator Parameters	33
7.3. Result	33
8. Decoder Programming Examples	35
A. Support	39
List of Figures	40
List of Tables	41

1. General

1.1. Welcome to the Decoder Debugger

The Decoder Debugger supports decoder development based on the *Decoder Description Language DDL* for go2signals systems.

The graphical user interface of these systems (SDA) already provides basic tools to create and modify decoders, however, the Decoder Debugger allows for detailed program analysis including the monitoring of variables, data input/output and the setting of breakpoints. Decoders can be tested and debugged in two different modes.


In Offline Mode, pure decoder functions can be investigated without any system overhead during runtime. The system configuration required is shown in Figure 17.

To observe the behavior interaction with demodulators and the automatic production control as well, run the debugger in Online Mode as graphical user interface of the APC program. This requires the system depicted in Figure 20.

In either mode, decoder programs can be executed and monitored in various single-step modes as well as continuously with or without breakpoints.

1.2. Software Environment

All functions described are embedded in the application "APC.exe", which also contains the complete signal processing environment of go2DECODE. During normal operation, APC runs as a background process without any visible user interface. However, a user interface for decoder debugging purposes may be activated during APC runtime via the SDA user interface. This mode allows for full control online debugging of decoders and automatic processing behavior. This <Decoder Debugger Online> is started via the SDA's <Extras> menu.

You can also start APC in sole decoder debugging mode without any signal processing functions. In this mode, only a decoder interpreter is activated, and controlled by a similar user interface. APC is started in this mode via the link <DecDebug> or the start icon  when running the <Decoder Debugger Offline>.

2. Decoder Adaptation and Development

Provided the source code for the supplied decoders is available, you may adapt or modify the decoders to suit your requirements. Additionally, new decoders can be created to execute other modems by use of the description language DDL (*Decoder Description Language*). DDL is a simple programming language developed specifically for signal decoding tasks.

2.0.1. Create Decoder

Figure 1 is an overview of all elements required to create and operate a decoder.

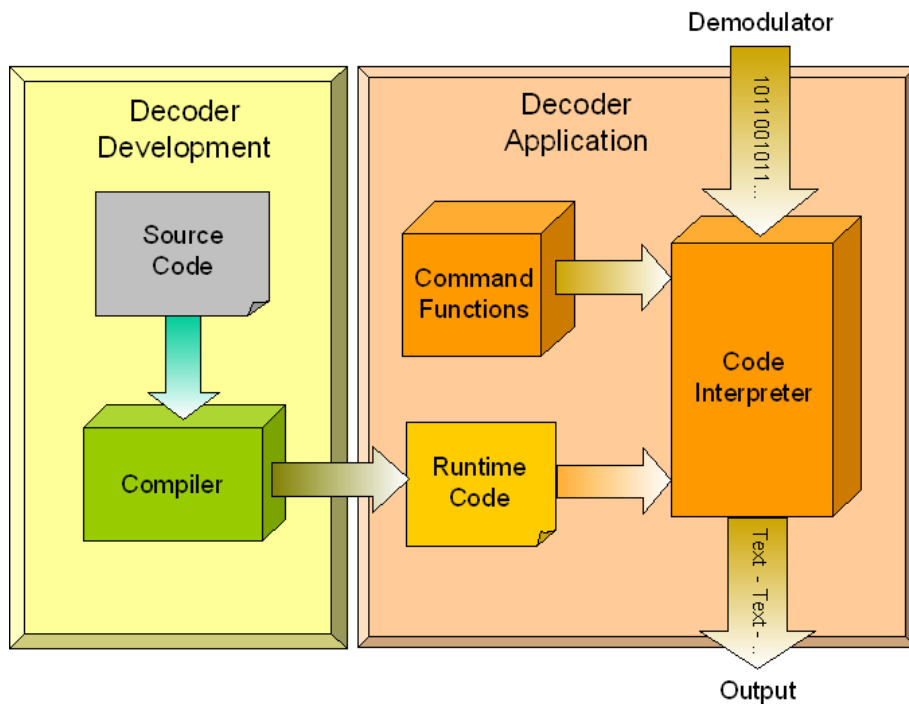


Figure 1: Overview of Decoder Creation

The decoder function is entirely defined in a source code, which can be created in basically any ASCII text editor. The source code comprises the command steps and the sequence in which these are to be processed during decoder application. A compiler translates this text into a code, which can be interpreted easily and quickly during the runtime of the decoder. The source code and decoder code are stored in files.

The code files created this way are used when integrating decoders in completed modems and when processing these modems.

2.0.2. Edit Decoder

2.0.2.1. Load Editor

To call the source codes of the decoders, use **<Edit description...>** in the **<Decod>** tab as shown in Figure 2. This is only possible if the decoder source code is available in your installation. Decoders requiring a special license option are not supplied with their source code.

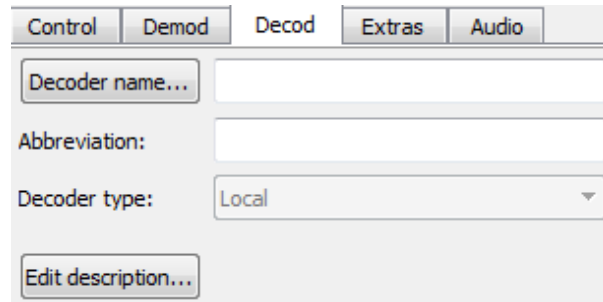


Figure 2: Load Decoder Editor

The editor window is displayed as shown in Figure 3, together with a description of the selected decoder. The various language elements are automatically displayed in different colors for improved overview. The color assignment is as follows:

Colors	Language Elements
Green	Comments
Red	Designators of program sections
Blue	Command functions and branch commands
Yellow	System variables
Black	Remaining text

Table 1: Decoder Editor Color Assignments

The editing and adapting functions are the same as in any standard text editor.

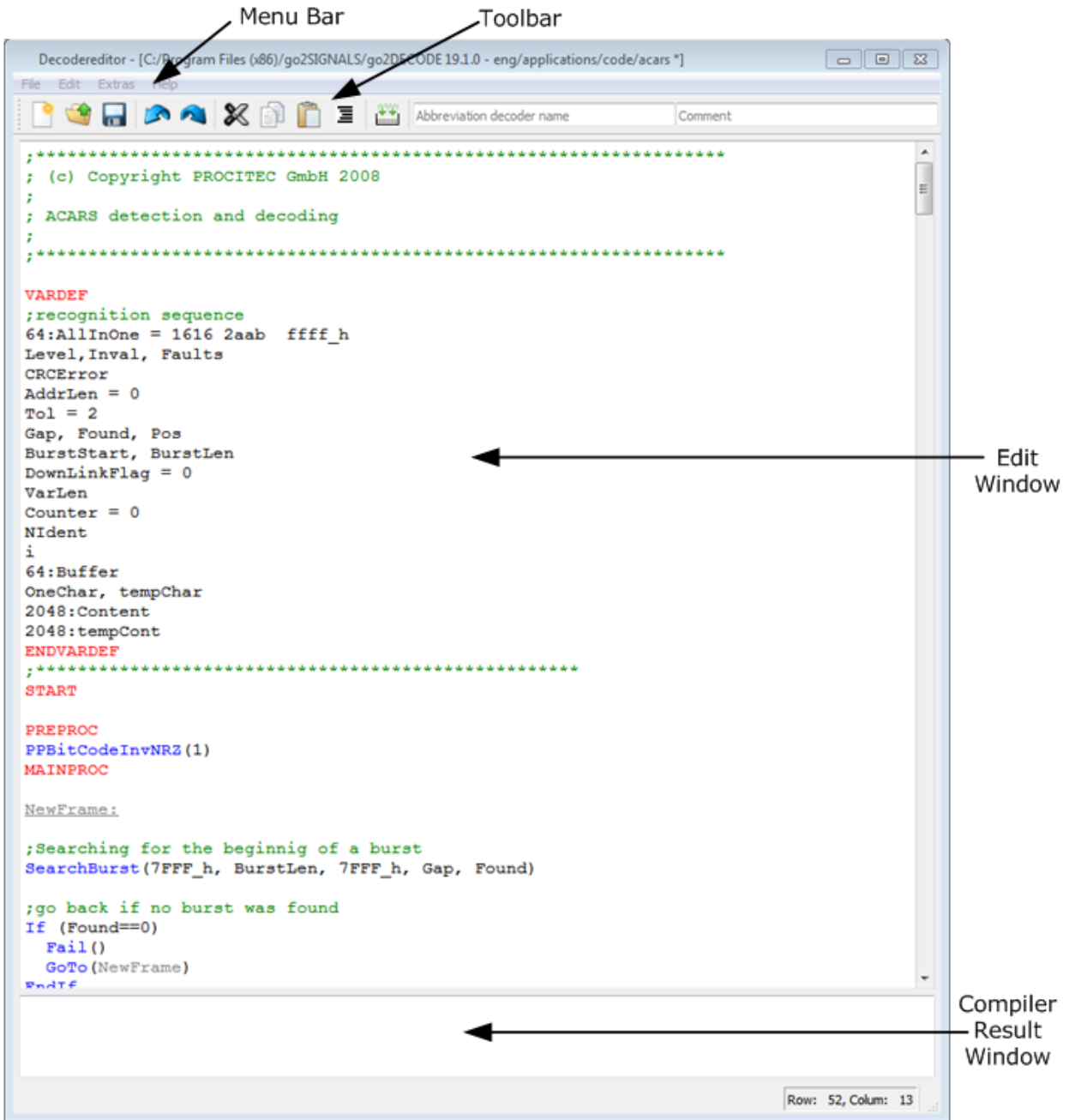


Figure 3: Decoder Editor

2.0.2.2. Menu Bar

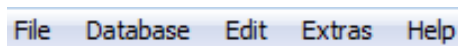


Figure 4: Menu Bar

The menu bar consists of five menus, featuring the following menu items:

Menu Item	Function
<File>	Management of decoder descriptions
<Database>	Connection to database
<Edit>	Editing functions and decoder creation
<Extras>	Automatic indentation, parameter information, and mass compilation
<Help>	Instruction Manual to Decoder Description Language DDL

Table 2: Decoder Editor Menu Items

2.0.2.2.1. File Menu

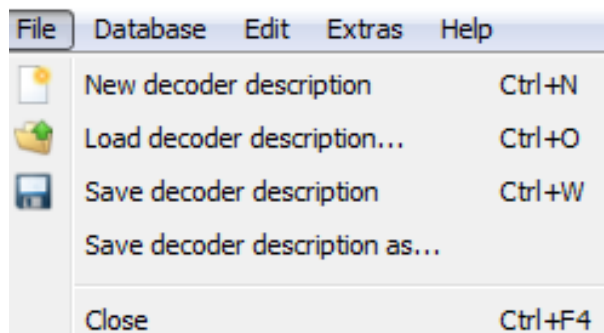


Figure 5: File Menu

Menu Item	Function
<New decoder description>	Remove all decoder descriptions previously displayed to release a new description
<Load decoder description...>	Load existing decoder description
<Save decoder description>	Save new / modified description
<Save decoder description as...>	Save decoder description using a new file name
<Close>	Close editor windows

Table 3: Decoder Editor File Menu Items

2.0.2.2.2. Edit Menu

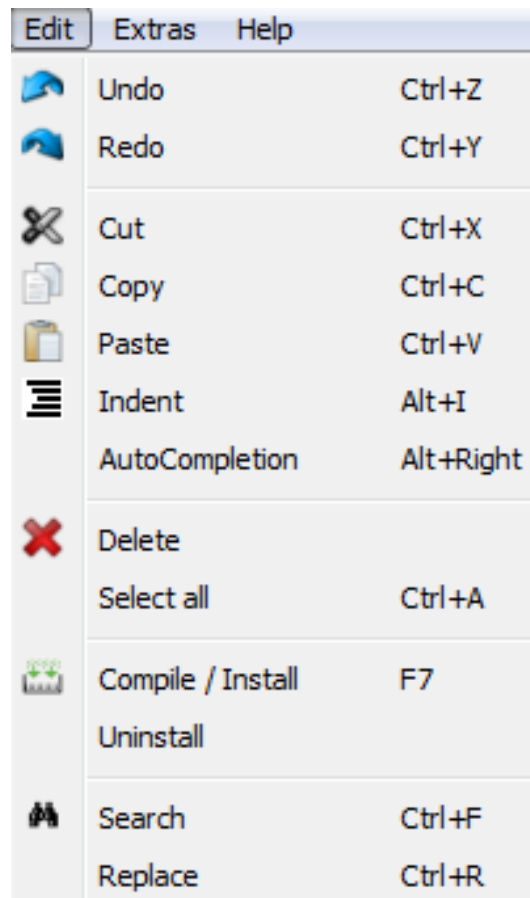


Figure 6: Edit Menu

<Edit> comprises the standard editor commands:

Menu Item	Function
<Undo>	Undo last change
<Redo>	Redo change undone
<Cut>	Cut highlighted text block (and copy to clipboard)
<Copy>	Copy highlighted text block to the clipboard
<Paste>	Insert clipboard contents at cursor position
<Indent>	Correct right and left indentation of highlighted text
<AutoCompletion>	Complete DDL command entry automatically
<Delete>	Delete highlighted block
<Select all>	Highlight the complete text
<Compile / Install>	Compile the edited text and create a code that is interpretable during the decoder runtime. The decoder code thus created is installed in the connected signal-processing channel
<Uninstall>	Remove decoder from the connected signal-processing channel

Menu Item	Function
<Search>	Search the entire text document for a specifiable text
<Replace>	Replace the specified text with another text

Table 4: Decoder Editor Edit Menu Items

2.0.2.2.3. Extras Menu

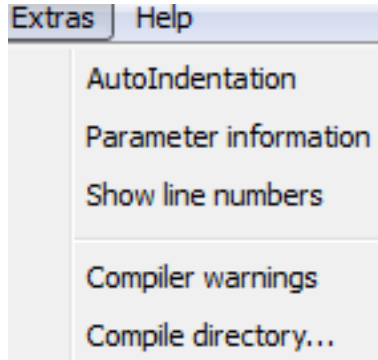


Figure 7: Extras Menu

<Extras> menu (see Figure 7) provides five items to facilitate the editing:

Menu Item	Function
<AutoIndentation>	Automatically insert as many blanks as in the previous line.
<Parameter information>	Show list of available parameters for valid DDL commands.
<Show line numbers>	In editor show column with line numbers.
<Compiler warnings>	During compilation show warnings besides errors.
<Compile directory...>	Compile all decoders in specific directory.

Table 5: Decoder Editor Extras Menu Items

2.0.2.2.4. Help Menu

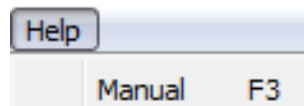


Figure 8: Help Menu

This menu features the item <Manual>, which serves to display the Instruction Manual to the Decoder Description Language *DDL*.

2.0.2.3. Toolbar

Specific functions can be executed via mouse click immediately on the toolbar.



Figure 9: Toolbar

The following functions are available:











Icon	Function
	New decoder description
	Load decoder description from file
	Save decoder description to file
	Undo
	Redo
	Cut
	Copy
	Paste
	Indent
	Compile

Table 6: Decoder Editor Toolbar Icons

2.0.2.4. Shortcuts for Decoder Creation

Shortcuts serve to quickly access specific functions in the creation of decoders.

Function	Shortcut
New decoder description	<Ctrl>+<N>
Load decoder description	<Ctrl>+<O>
Save decoder description	<Ctrl>+<W>
Undo	<Ctrl>+<Z>
Redo	<Ctrl>+<Y>
Cut	<Ctrl>+<X>
Copy	<Ctrl>+<C>
Paste	<Ctrl>+<V>
Indent	<Alt>+<I>
Select all	<Ctrl>+<A>

Function	Shortcut
Search	<Ctrl>+<F>
Replace	<Ctrl>+<R>
Open context-sensitive DDL help	<F3>
Compile	<F7>

Table 7: Decoder Editor Shortcuts

2.0.2.5. Context-Sensitive Help

This function serves to display the documentation on a valid DDL command. To do so, position the cursor on a DDL command in the decoder editor and press <F3>. Subsequently, the software opens the documentation and searches the DDL Operating Instructions for the current DDL command. In case the current text string is no valid DDL command, the search will not produce any result.

Note: The precondition for the correct function of the context-sensitive help is an existing DDL Operating Instructions file.

2.0.2.6. Automatic Command Completion

When entering a DDL command, the software can complete the current text entry automatically into a DDL command. Activate this function by means of the shortcut <Alt>+<→>. If the entry unequivocally matches a DDL command, the missing characters are inserted immediately upon activation of <Alt>+<→>.

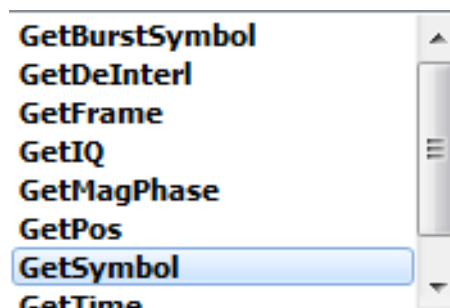


Figure 10: Automatic Command Completion

If the current text string matches several valid DDL commands, a list box is displayed showing the commands in question (see Figure 10). The list box shows a maximum of ten possible completions. Select the desired completion using the arrow keys or the mouse. Insert the entry selected in the list box at the current text position by pressing <Return> or double clicking the desired item.

Exit the list box at any time via <Esc> or clicking any position outside the list box on screen.

If no valid DDL commands match the current entry, you will see an alert message (Completion not possible) on activating <Alt>+<→>. The text string to complete must consist of at least two characters.

2.0.2.7. Automatic Indentation

On pressing <Return>, this function inserts the same number of blank spaces as in the previous line. Additionally, it will insert two blanks after an *If*, *Case*, *For*, *Switch* or *While* command.

Activate and deactivate the automatic indentation on the <Extras> menu. The setting remains unchanged on exiting the program and still be active next time you start the decoder editor.

2.0.2.8. Show Parameter Information

This function shows the list of available parameters for a valid DDL command when entering the "(" character. Output parameters are displayed in blue, optional parameters are in Italics. During the input, the current parameter is shown in bold and underlined characters (see Figure 11).

ValPattern, *CarePattern*, *Repeat*, *Tol*, *Faults*, *GapLimit*, *Gap*, *Found*

Figure 11: Decoder Status Bar

The parameter information remains on screen until you enter the character ")" in the current line; or when clicking anywhere else on screen or when entering something else which is no valid DDL command or when scrolling the text in the editor pane.

Activate and deactivate the parameter information on the <Extras> menu. The setting remains unchanged on exiting the program and still be active next time you start the decoder editor.

2.0.2.9. Compile Directory

This function serves to compile all decoders in a specific directory. On activation of this item on the <Extras> menu, the program shows a dialog box for selection of the desired directory.

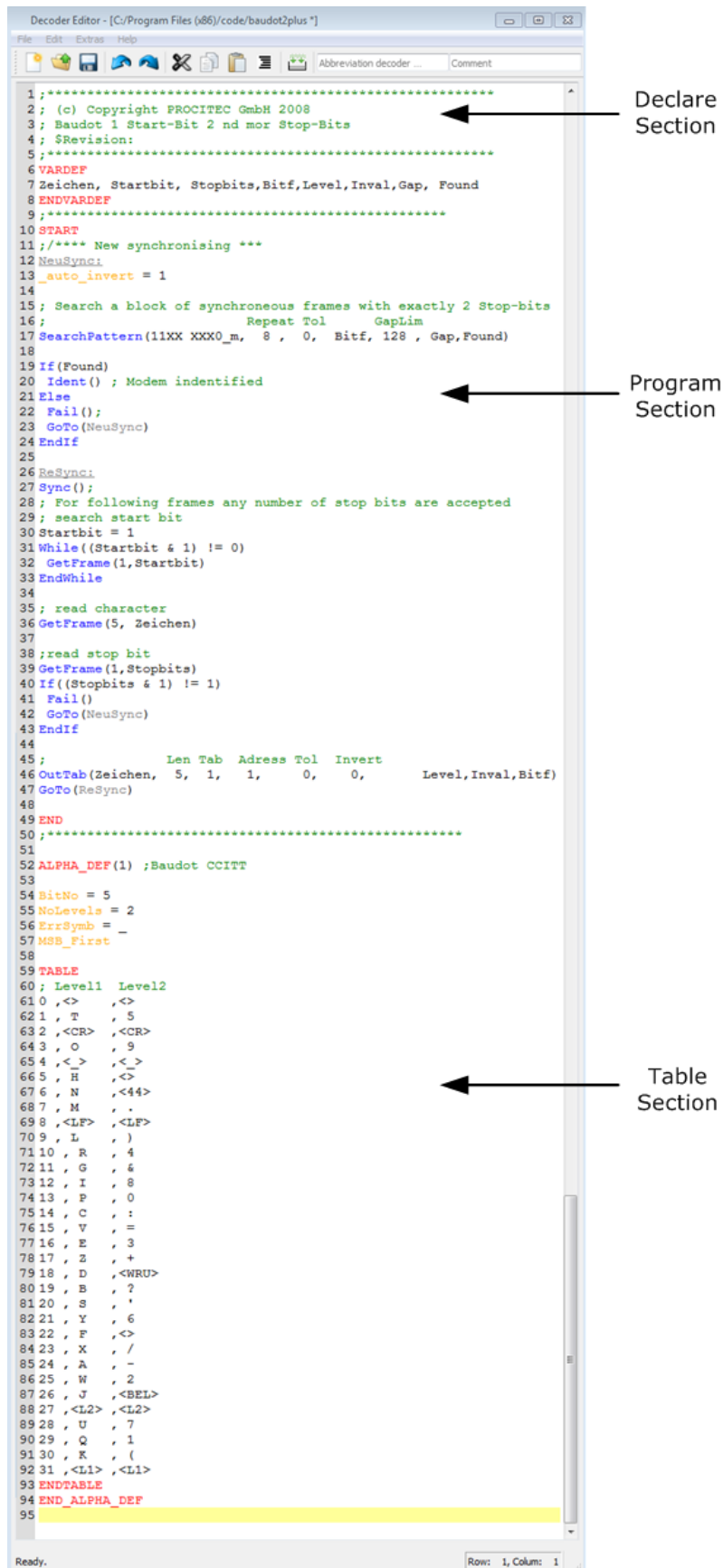
2.0.3. Decoder Source Code Structure

The Decoder Description Language is the basis for the source code for the description of decoders. Please consult the document Decoder Description Language DDL for a more detailed description of the structure and the various command elements. To view this document, use <Help> menu or the <F2> hot key. The descriptions below will merely provide a rough and initial overview.

In general, the syntax of the decoder description corresponds to that of a simple programming language. Programs always begin with a declaration part which defines the variables used, followed by the actual description of the program flow, which uses both fundamental and very specific commands allowing for implementation of more complex decoder functions with only one command line. Schematic assignments like alphabet encoding may be defined in separate tables. Reference to these tables can be made via specific commands in the course of the program flow.

Figure 12 illustrates the basic structure of a description, using a simple decoder as an example. Every program is adapted to the general basic data flow as shown in Figure 13.

First, the incoming data stream is stored automatically in the input buffer where it is possible to search for specific data patterns or characteristics to identify the modem or to configure the start synchronization. Starting with the positions detected this way; the data stream can be read out and processed in steps. Optionally, pre-processing operations, i.e. the modification of the incoming data stream, can be carried out before saving the data in the input buffer.



```

1;*****
2; (c) Copyright PROCITEC GmbH 2008
3; Baudot 1 Start-Bit 2 nd mor Stop-Bits
4; $Revision:
5;*****
6 VARDEF
7 Zeichen, Startbit, Stopbits, Bitf, Level, Inval, Gap, Found
8 ENDVARDEF
9;*****
10 START
11;**** New synchronising ****
12 NeuSync:
13 _auto_invert = 1
14
15; Search a block of synchronous frames with exactly 2 Stop-bits
16; Repeat Tol GapLim
17 SearchPattern(11XX XXX0_m, 8, 0, Bitf, 128, Gap, Found)
18
19 If(Found)
20 Ident(); Modem identified
21 Else
22 Fail();
23 GoTo(NeuSync)
24 EndIf
25
26 ReSync:
27 Sync();
28; For following frames any number of stop bits are accepted
29; search start bit
30 Startbit = 1
31 While((Startbit & 1) != 0)
32 GetFrame(1, Startbit)
33 EndWhile
34
35; read character
36 GetFrame(5, Zeichen)
37
38; read stop bit
39 GetFrame(1, Stopbits)
40 If((Stopbits & 1) != 1)
41 Fail()
42 GoTo(NeuSync)
43 EndIf
44
45; Len Tab Adress Tol Invert
46 OutTab(Zeichen, 5, 1, 1, 0, 0, Level, Inval, Bitf)
47 GoTo(ReSync)
48
49 END
50;*****
51
52 ALPHA_DEF(1); Baudot CCITT
53
54 BitNo = 5
55 NoLevels = 2
56 ErrSymb = -
57 MSB_First = -
58
59 TABLE
60; Level1 Level2
61 0, <>, <>
62 1, T, 5
63 2, <CR>, <CR>
64 3, O, 9
65 4, <_>, <_>
66 5, H, <>
67 6, N, <44>
68 7, M, .
69 8, <LF>, <LF>
70 9, L, )
71 10, R, 4
72 11, G, &
73 12, I, 8
74 13, P, 0
75 14, C, :
76 15, V, =
77 16, E, 3
78 17, Z, +
79 18, D, <WRU>
80 19, B, ?
81 20, S, '
82 21, Y, 6
83 22, F, <>
84 23, X, /
85 24, A, -
86 25, W, 2
87 26, J, <BEL>
88 27, <L2>, <L2>
89 28, U, 7
90 29, Q, 1
91 30, K, (
92 31, <L1>, <L1>
93 ENDTABLE
94 END_ALPHA_DEF
95

```

Annotations in the image:

- Declare Section:** Points to lines 1-5 (header and variable definitions).
- Program Section:** Points to lines 10-50 (main logic and control flow).
- Table Section:** Points to lines 59-93 (the mapping table).

Figure 12: Example of a Decoder Program

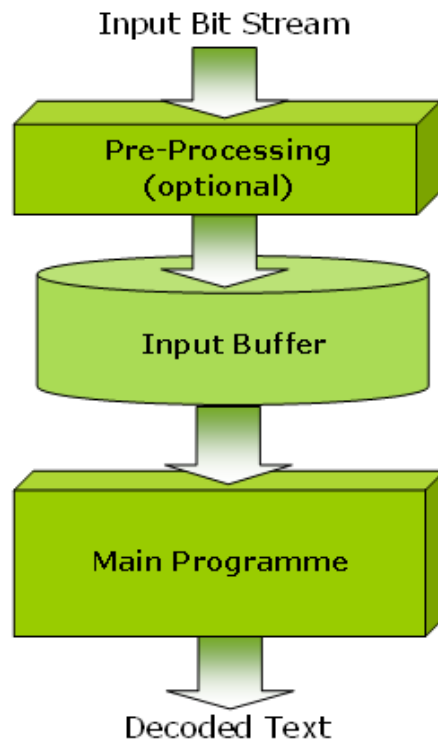


Figure 13: Decoder Data Flow

Most of the programs apply a choice of the following command types:

Search Commands (starting with Search...)

These commands search for special patterns and identification characteristics within the input buffer which is refilled after each call. The search can be made with error tolerance. These commands are required for both the identification of an encoding and the synchronization of an appropriate start position.

Examples:

Command	Functions
SearchPattern	Search for specific bit patterns
SearchSymbolTab	Search for symbols defined in a table
SearchInterlSymbolTab	Corresponding search of interleaved symbols
SearchBurst	Search for the start of a burst
SearchPolynom	Search for an output bit sequence of a feedback shift register LFSR
SearchVectorPatternMatch	Search for patterns consisting of multi-order input symbols (for example PSK8 sequences).

Table 8: DDL Search Commands

Read Commands (starting with Get...)

These commands initiate the reading of data blocks from the input buffer in variables:

Examples:

Command	Functions
GetFrame	Read specific quantity of bits
GetDelInterleave	Read specific quantity of bits according to a definable interleaving pattern
GetSymbol	Read specific quantity of multi-order input symbols

Table 9: DDL Read Commands

Frame Fragmentations and Reformatting

Distributed or interleaved bit sequences can be composed in various ways.

Examples:

Command	Functions
Extract	Extract bit frame from variable, with or without bit reversal
ExtractInterl	Extract interleaved bit frame
ExtractPattern	Extract word distributed in freely definable bit positions
Destuff	Delete stuffing bits
Join	Join two bit sequences

Table 10: DDL Bit Manipulation Commands

Check and Correction Methods

Examples:

Command	Functions
CheckCRC	Execution of Cyclic Redundancy Checks
CorrectExtGolay	Error correction of an Extended Golay Code
TestPolynom	Test whether a bit sequence was created by a linear feedback shift register (LFSR)
Weight	Count the quantity of ones in a test word
ViterbiHDD	Decode and correct convolutional code according to Viterbi Hard Decision Algorithm.
IsTabSymb	Check whether a bit sequence consists of valid symbols of a symbol table

Table 11: DDL Check and Correction Commands

Operators (=, +, -, /, *, &, |, && ...)

Arithmetic, binary and logical operators, as well as bracket operators, can be nested deliberately for use in assignment equations or parameter assignments. The syntax follows the structure of the programming language C.

Branch Commands (If, While, For, GoTo)

Create conditional or unconditional branches and loops.

Output Commands (starting with Out...)

This group of commands serves to output results to specific result addresses (displays, database categories, etc.). To a certain extent, these commands can also convert table code in one step.

Examples:

Command	Functions
OutTab	Decoding of a bit field according to a symbol table (for example character alphabets) and output
OutTabHuffman	Same function for Huffman alphabets (unequal symbol lengths)
OutText	Output of any additional text
OutVal	Output of a numeric value
OutTimeStamp	Output of a time stamp to a specific bit of the input buffer

Table 12: DDL Output Commands

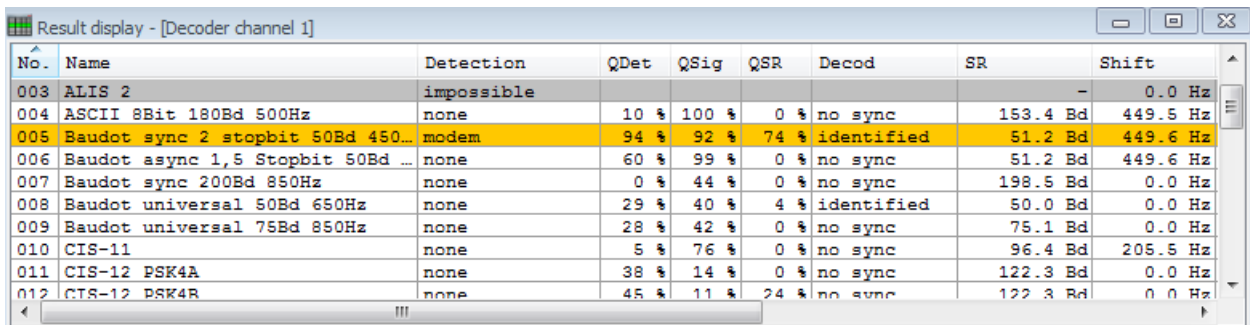
Control Commands

These commands serve to provide the calling production Automat with messages like identification of the correct modem, defined loss of identification, or status of process ability without any identification statement. Using these commands is the precondition for flawless automatic modem identification. The status at the time can be verified in the result window during modem search (see Figure 14).

Examples:

Command	Functions
Sync	The signal can be processed
Ident	The modem has been identified but on condition that the production Automat accepts the signal quality.
Access	The modem has been identified clearly enough to omit the quality check.
Fail	The modem is not (no longer) available

Table 13: DDL Output Commands



No.	Name	Detection	QDet	QSig	QSR	Decod	SR	Shift
003	ALIS 2	impossible						0.0 Hz
004	ASCII 8Bit 180Bd 500Hz	none	10 %	100 %	0 %	no sync	153.4 Bd	449.5 Hz
005	Baudot sync 2 stopbit 50Bd 450...	modem	94 %	92 %	74 %	identified	51.2 Bd	449.6 Hz
006	Baudot async 1,5 Stopbit 50Bd ...	none	60 %	99 %	0 %	no sync	51.2 Bd	449.6 Hz
007	Baudot sync 200Bd 850Hz	none	0 %	44 %	0 %	no sync	198.5 Bd	0.0 Hz
008	Baudot universal 50Bd 650Hz	none	29 %	40 %	4 %	identified	50.0 Bd	0.0 Hz
009	Baudot universal 75Bd 850Hz	none	28 %	42 %	0 %	no sync	75.1 Bd	0.0 Hz
010	CIS-11	none	5 %	76 %	0 %	no sync	96.4 Bd	205.5 Hz
011	CIS-12 PSK4A	none	38 %	14 %	0 %	no sync	122.3 Bd	0.0 Hz
012	CIS-12 PSK4B	none	45 %	11 %	24 %	no sync	122.3 Bd	0.0 Hz

Figure 14: Result Display on Calling the Command Ident

In some cases decoders require the processing of the incoming bit stream before saving it to the input buffer. These commands must be listed before the main section of the program.

Pre-processing Commands

Examples:


Command	Functions
PPInvert	Invert every input bit
PPDescramble	Execute a descrambling function for any specified polynomial
PPBitCodeBIPH	Reverse BIPH bit encoding
PPBitCodeNRZ	Reverse NRZ bit encoding
PPBitCodeManch	Reverse Manchester bit encoding
PPConvertIcon	Convert multi-order input symbols according to a table
PPSymbolBitReversal	Reverse bit order of a multi-order input symbol

Table 14: DDL Pre-Processing Commands

Recommendation: Use a simple, comparable and working decoder as a model and modify this decoder systematically.

2.0.4. Compile and Operate New Decoders

2.0.4.1. Start Compiler

Complete decoder descriptions are compiled via the icon  on the editor toolbar or via the <Edit> <Compile / Install> menu (see Figure 6). The result is displayed in the pane below the edited text. Successful compilation is indicated by the message *Compilation successful* in the final row. In this case, the executable decoder code has been created and is available for loading.

In case of incorrect source text, an error message will be shown which also indicates the row number in question. A mouse click on the error row indicated will automatically position the cursor in this row in the text box. Any error messages at this point merely refer to incorrect syntax and the formal integrity of the program.

```

Compiler: test started.

Compiler.exe (C) Copyright PROCITEC GmbH 2018
Version 19.1.0 1811081538PROCITEC-INTERNE-TESTVERSIONR.64W001900

Compile Pass 1: C:/Users/anyuser/somesubdir/test.txt
List: C:/Users/anyuser/somesubdir/test.list
wait...

Compile Pass 2: C:/Users/anyuser/somesubdir/test.sr2
Code: C:/Users/anyuser/somesubdir/test.bin

Compilation successful

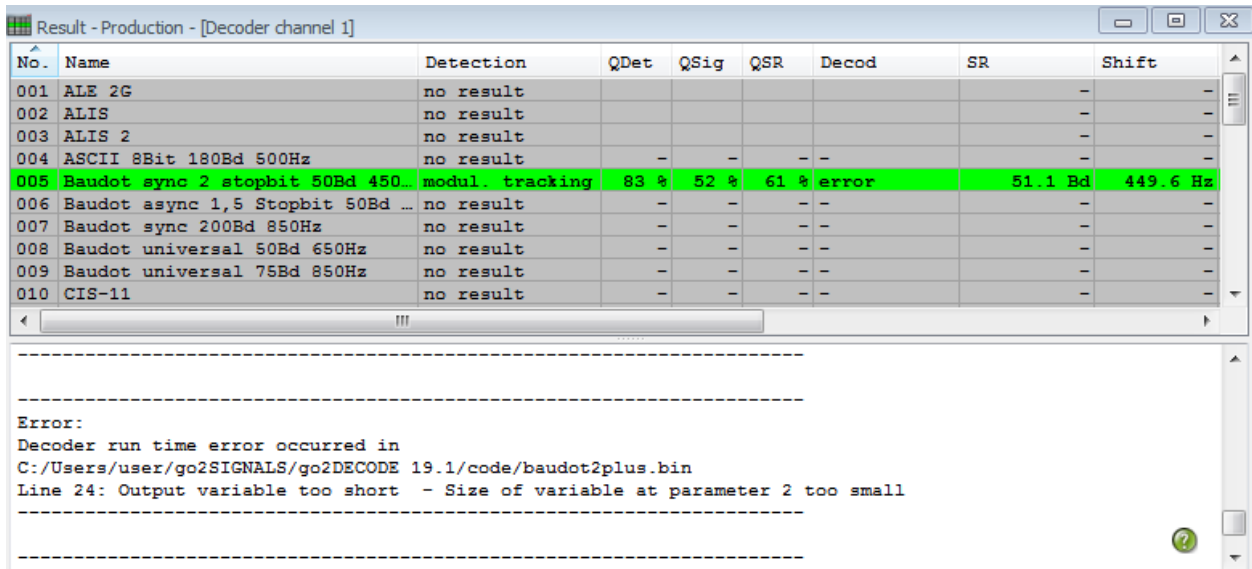
Ready. Row: 1, Colum: 1
    
```

Figure 15: Message on Successful Compilation

The proper operation of a program, however, cannot be verified until the new decoder has been started. The new decoder code is displayed in the decoder selection list. The code has been previously saved to a file with the extension “.bin”. For initial testing, we recommend to start a manual production process with a suitable test signal.

Runtime errors may occur in this process. These errors will cause error messages, which are displayed in the column *Decod* in the result display. To obtain a detailed error message, activate the check box *XML Tag Filter* on the <Extras> tab of the result display.

Runtime errors are caused by errors that were impossible to detect in the compilation process. Such errors are e.g. inadmissible parameter values, exceeding of runtime due to endless loops, inappropriate input formats due to incorrect demodulators, or internal buffer overflow.



No.	Name	Detection	QDet	QSig	QSR	Decod	SR	Shift
001	ALE 2G	no result						-
002	ALIS	no result						-
003	ALIS 2	no result						-
004	ASCII 8Bit 180Bd 500Hz	no result	-	-	-	-	-	-
005	Baudot sync 2 stopbit 50Bd 450...	modul. tracking	83 %	52 %	61 %	error	51.1 Bd	449.6 Hz
006	Baudot async 1,5 Stopbit 50Bd ...	no result	-	-	-	-	-	-
007	Baudot sync 200Bd 850Hz	no result	-	-	-	-	-	-
008	Baudot universal 50Bd 650Hz	no result	-	-	-	-	-	-
009	Baudot universal 75Bd 850Hz	no result	-	-	-	-	-	-
010	CIS-11	no result	-	-	-	-	-	-

```

-----
Error:
Decoder run time error occurred in
C:/Users/user/go2SIGNALS/go2DECODE 19.1/code/ baudot2plus.bin
Line 24: Output variable too short - Size of variable at parameter 2 too small
-----
    
```

Figure 16: Display of Runtime Error

The row number indicated in the error message (“Line 24” in Figure 16) refers to the specific position in the source code where an inadmissible condition has been identified and the program has been aborted.

3. Offline Mode

3.1. Offline Operating Concept

In this mode, the decoder is operated within an isolated environment, i.e. without being affected by APC control during runtime. This operating platform is provided for initial decoder development operations. It is easy to handle and decoders most often can be run faster than under real-time conditions, which is a helpful feature with time-consuming slow baud rates. The basic approach for this mode can be described as follows.

Under real operating conditions, the decoder will be fed by the demodulator output. For debugging, the decoder will now access a recording of this output. Therefore, in a first step, it is necessary to produce a record file as described in chapter Producing a Demodulator Output Record on page 19.

The recording will contain the complete set of information produced for decoder input during recording runtime, such as bit stream, symbol and channel arrangements, burst information, time of arrival etc. To start debugging, load only this record file in connection with the compiled decoder including the source code.

The <Decoder Debugger Offline> mode, however, does not allow for monitoring any interactions with external APC functions such as:

- Manipulation of demodulation parameters initiated by the decoder
- The automatic modem detection process
- The effect of decoder results for the go2DECODE system

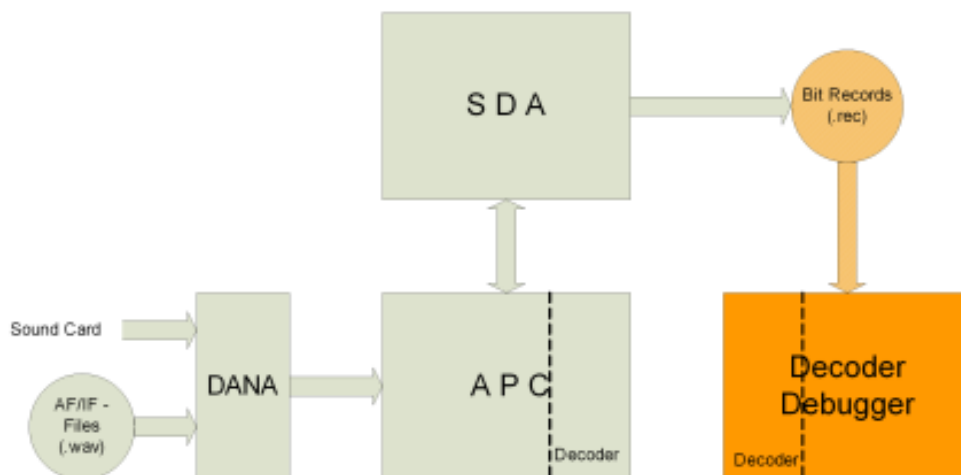


Figure 17: Decoder Debugger – Offline Operating Environment

3.2. Decoder Debugger - Offline Main Window

The <Decoder Debugger Offline> user interface elements are shown in Figure 18, followed by a brief description.

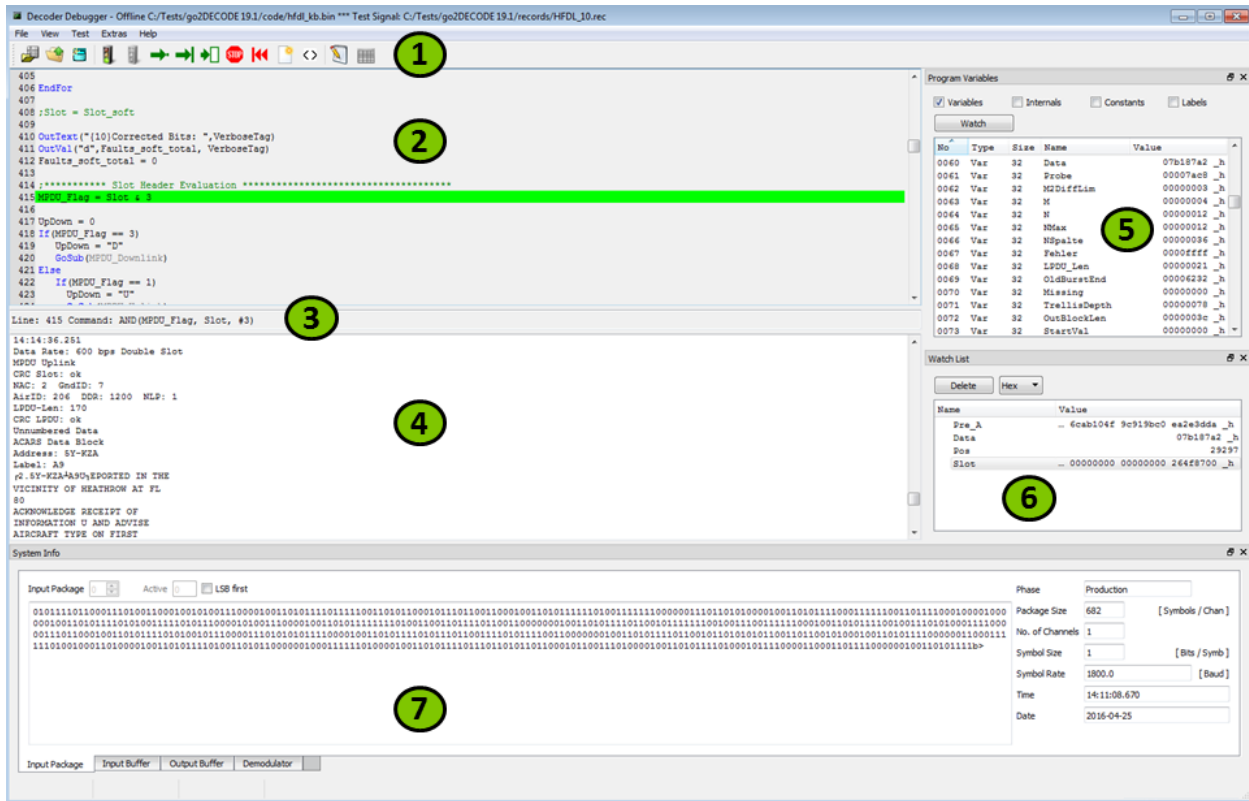


Figure 18: Decoder Debugger – Offline Main Window


1. Menu Bar and Toolbar
Central functions can be accessed via the menu bar. For most of these functions, toolbar icons are provided as well.
2. Source Display
The source code of a loaded decoder is displayed in this window. Breakpoints and the subsequent program line are highlighted in different colors. This display is read-only. Use the editor for modifying source codes.
3. Command Line Display
Each source code corresponds to a single, or a set of, runtime commands which are composed during compilation. The next runtime command is displayed in this line.
4. Result Display
The decoding results are displayed in this pane. Results may be raw, XML text or filtered.
5. Variables List
This pane displays all program variables and their current values.
6. Watch List
This pane may contain a selected subset of the Variables List. Values can be indicated in different formats.
7. Buffer and Parameter Display Tabs
The panes on these tabs provide information on:
 - Last input package received from demodulator

- Input buffer with current read pointer indication
- Output buffer containing current output command results
- Demodulator parameters
- Additional display for selectable long variables

3.3. Producing a Demodulator Output Record

Demodulator output records can be produced via the go2DECODE user interface SDA. These records are images of the real-time decoder input data streams.

The correct demodulator has to be prepared and must be run in manual production mode, i.e. the automatic signal processing (button <Automat>) must be switched off. No decoder is required, but allowed while recording. For details on the running of modems, please refer to the go2DECODE Instruction Manual.

The recording can be started by the icon . It will run until the modem is stopped. The selected record file name can be used as Decoder Debugger data input.

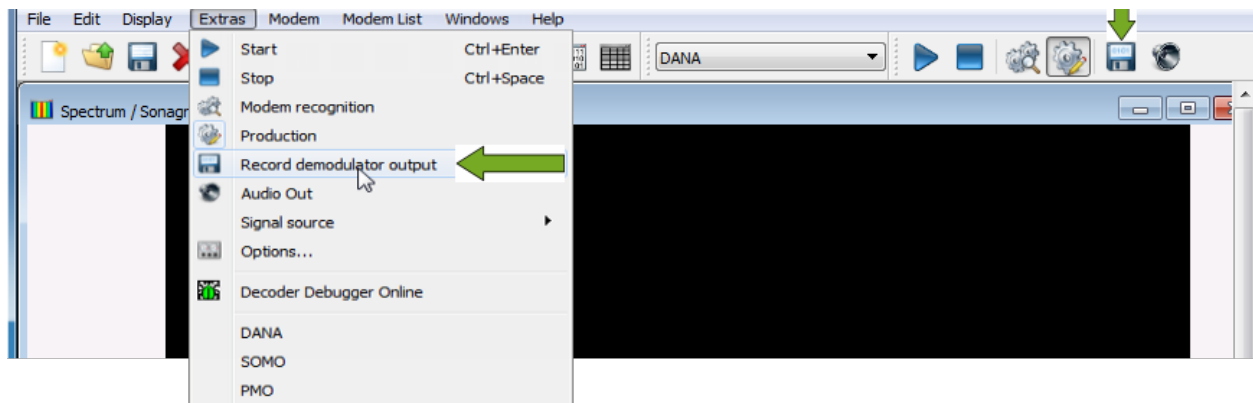


Figure 19: Starting Demodulator Output Records

3.4. Loading Record File and Decoder

Compiled decoders and recorded demodulator outputs are required for decoder debugger operation. If only source code is available, it must be loaded via the decoder editor and compiled first. Existing decoders are loaded via

- <File><Open><Decoder>, <Ctrl>+<O>, 

The displayed file browser offers all files with the extension "*.bin", i.e. compiled decoder codes. If the corresponding source code (extension "*.txt") is found in the same directory, it will automatically be loaded as well. An alert message is displayed if the source code has been changed since the last compilation.

Pre-recorded data are loaded via

- <File><Open><Data>, <Ctrl>+<D>, 

The displayed file browser will offer all files with the extension "*.rec" or an alternative (obsolete) format "*.bit".

4. Online Mode

4.1. Operating Concept of Online Debugging

All the interactions between the decoder and other go2DECODE functions can be examined online during normal operation. In this mode, the decoder runs in its original environment but is observed and controlled by the APC's debugging user interface (see Figure 20). In order to have a running system, the applications SDA and DANA must be started and operated as well, i.e. the system can be run as the original go2DECODE via SDA as long as there is no interrupt by the decoder debugger user interface. There are no restrictions with regard to the signal source, but loss-free single step operation is only possible with file source input.

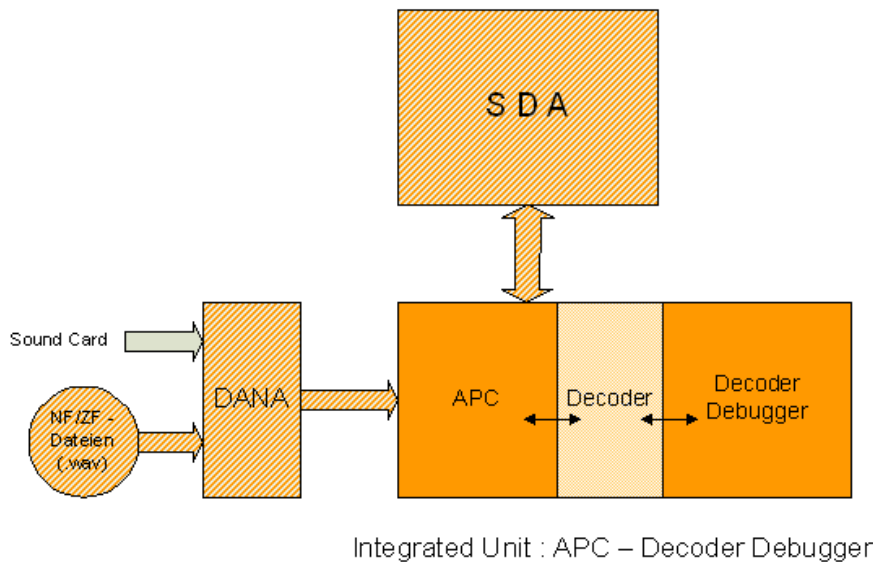


Figure 20: Online Debugging Concept

Start the go2DECODE system as usual. The decoder and modem to be observed must be present in the loaded modem type list. Start the production manually or automatically. From the debugger user interface, now select the complete modem from the modem type list rather than just the decoder. This list is displayed in an additional window in the debugger user interface, representing a copy of the SDA modem type list.

Subsequently, debug the decoder or modem, resp., during automatic production, both in search phase and in production phase. Define breakpoints to be active in both search and production phase, or in production phase only. On reaching the breakpoints, single stepping as well as system monitoring can be done the same way as in pure <Decoder Debugger Offline> mode, and the effects of decoder commands, editing of demodulator parameters or automatic production control can be observed as well.

4.2. Selecting the Decoder Debugger – Online User Interface

Open the online debugging user interface via

- menu item <Extras><Decoder Debugger Online> or press .

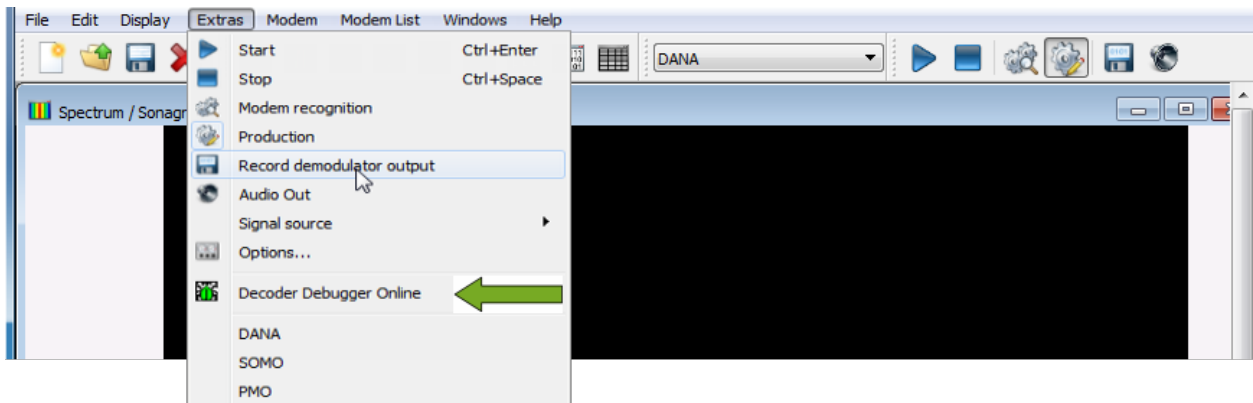


Figure 21: Select Decoder Debugger

The following main window will appear, a slightly modified window compared to one in <Decoder Debugger Offline> mode:

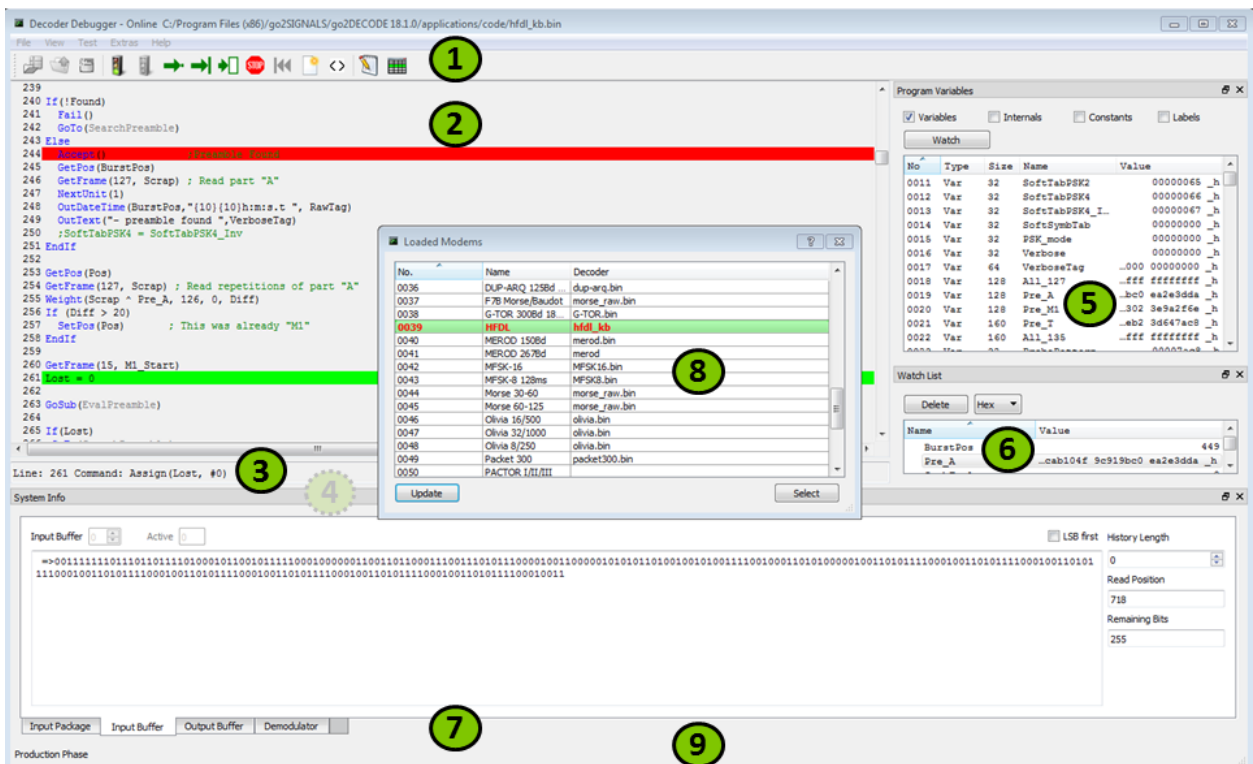



Figure 22: Decoder Debugger - Online Main Window

1. Menu Bar and Toolbar
Central functions can be accessed via the menu bar. For most of these functions toolbar icons are provided as well.
2. Source Display
The source code of a loaded decoder is displayed in this window. Breakpoints and the subsequent program line are highlighted in different colors. This display is read-only. Use the editor for modifying source codes.

3. Command Line Display
Each source code corresponds to a single, or a set of, runtime commands which are composed during compilation. The next runtime command is displayed in this line.
4. Result Display
missing
5. Variables List
This pane displays all program variables and their current values.
6. Watch List
This pane may contain a selected subset of the Variables List. Values can be displayed in different formats.
7. Buffer and Parameter Display Tabs
The panes on these tabs contain:
 - Last input package received from demodulator
 - Input buffer with current read pointer indication
 - Output buffer containing current output command results
 - Demodulator parameters
 - Additional display for selectable long variables
8. List of Loaded Modems
This list corresponds to the modem type list loaded via the application SDA. Once these modems are loaded into the APC (now as part of the Decoder Debugger), they will be displayed in this list. Modems can be marked, indicating their current operating status.
Bold: The modem/decoder is selected for monitoring
Yellow background: The modem is processed in search mode
Green background: The modem is processed in production mode
Red characters: Processing has stopped at a breakpoint within this modem/decoder
9. Status Fields
These fields indicate the operating status of the internal APC, and the connection status to the external applications SDA and DANA.

4.3. Selecting a Decoder for Debugging

As the decoder will be selected in conjunction with a complete modem via the modem list, remember to make the following preparations before selecting the decoder:

- Start SDA if it is not running. Successful connection to the SDA will be confirmed in the status fields.
- Load a modem type list via SDA containing the decoder of interest. On successful loading, a copy of this modem will be visible above the main window.
- If the list is hidden, select:
<View><Modem List> or press 

Press the button <Update> to ensure the latest changes are updated as well.

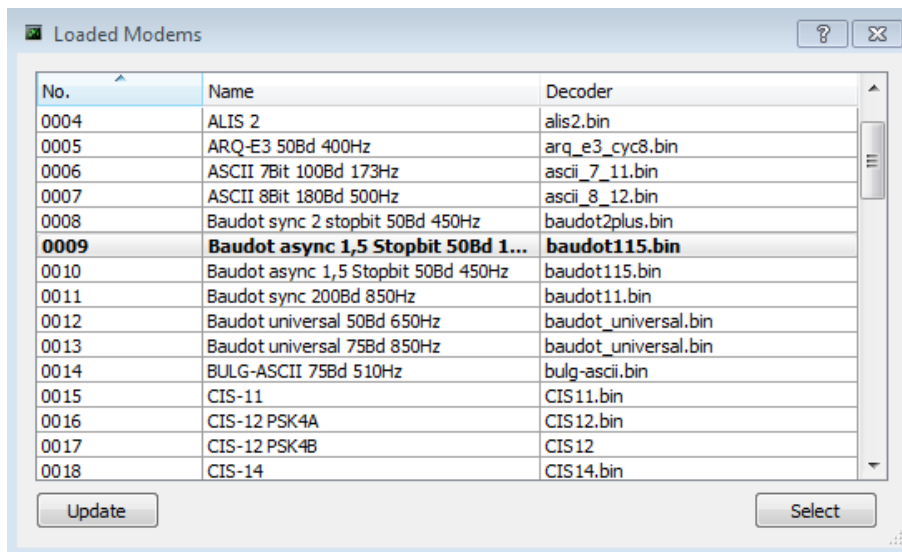


Figure 23: Loaded Modems List


To select a decoder as part of a modem, use <Select> or just double click the respective modem. As a result, the decoder source code will be shown in the source display.

Start the production via SDA as in the standard go2DECODE environment. The selected decoder can be accessed by setting breakpoints in any code line in the debugger source display. Subsequently, apply the debugging functions described in the next chapter.


5. Running and Analyzing Decoder Programs

5.1. Free Run

Decoders are started and executed continuously via

- **<Test><Run>**, **<F5>** or press 

To stop the running decoder before the end of input data, use

- **<Test><Hold>**, **<Esc>** or press 

5.2. Breakpoints

Breakpoints can be set in any source code line. Either double click the desired line number or select any valid code line (no blank or comment lines) and use

- **<Test><Breakpoint>**, **<F9>** or press 

The decoder application will run until it reaches the selected line. It can be continued in a stepping mode or continuous mode. Set as many breakpoints as required. To remove a single breakpoint, repeat the procedure in the same line.

In *Online* mode, breakpoints can be set in more than one decoder. They will be active whether the decoder is currently selected or not. Also in *Online* mode, you may specify whether the breakpoint shall be valid in production mode only or also during the search phase in automatic production, by activating the checkbox before **<Test><Breakpoints incl. Search>**.

Alternatively, the decoder program can run up to a dedicated line by setting the cursor to this line and selecting **<Test><Run to Cursor>**.

To remove all breakpoints, either in the current decoder or all breakpoints in all decoders, enter

- **<Test><Delete All Breakpoints><In this Decoder>** or **<Shift> + <F9>**.

Alternatively select


- **<Test><Delete All Breakpoints><In all Decoders>** or **<Ctrl> + <Shift> + <F9>**.

5.3. Stepping Modes

Once a decoder is stopped, i.e. at the beginning or at a breakpoint, it can be continued in three different stepping modes.


5.3.1. Single Line

The *Single Line* command will process the code covered by the next source code line. Select

- <Test><Single Line>, <F10> or press 


5.3.2. Single Step

The *Single Step* command will process the next step compiled and displayed in the command line. Select

- <Test><Single Step>, <F11> or press 


5.3.3. Single Package

Decoder input data is received in packages of various sizes. The sizes are determined by the demodulator. New packages are requested by the decoder whenever this is required in order to continue. Consequently, this function will continue until the next input package is requested. Select

- <Test><Single Package>, <Shift> + <F10> or press 


5.4. Restarting the Decoder

This function is available in Offline Mode. The decoder and input data are reset to their start condition by the command

- <Test><Reset>, <F12> or press 

5.5. Clear Result

This function will clear the result display which is used in Offline Mode. Select

- <Test><Clear Result>, <F8> or press 

6. Monitoring Variables

Variables can be monitored each time the run is stopped due to breakpoints or step commands.

6.1. Variables List

The Variables List contains all variables, labels and constants used in the program:

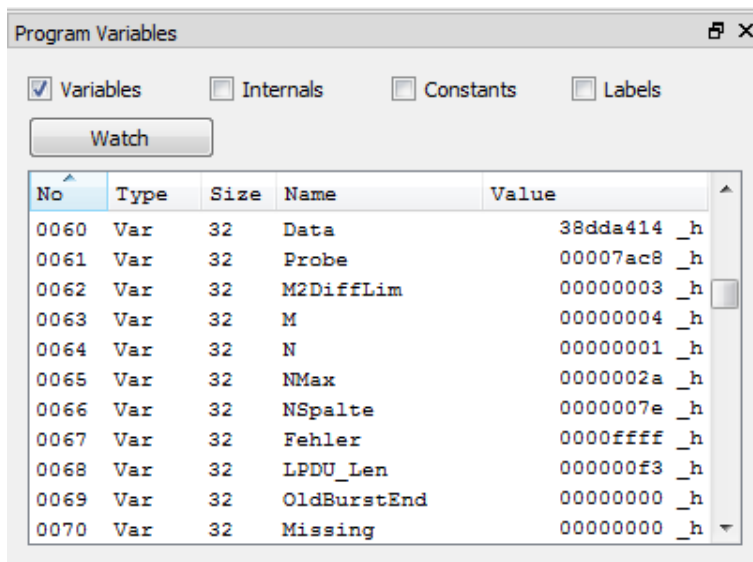


Figure 24: Variables List

Each variable is described by the following attributes:

Attribute	Description
No	Variable number as used in the compiler listing (i.e. file with extension .list produced during compilation)
Type	Variable type as described in chapter Variable Types
Size	Size of variable in bits
Name	Variable name
Value	Variable value

Table 15: Variables Attributes

6.1.1. Variable Types

There are the following types of variables:

- Declared program variables, shown as 'Var'.
- Internal variables, shown as 'Intern'. These variables are used either for system control or as auxiliary variables, generated during compilation, e.g. resolving nested operations.
- Constants, shown as 'Const', as used in the source code.
- Labels as used in the source code or generated during compilation, e.g. resolving branch constructions. The values correspond to program step numbers used internally.

6.1.2. Selecting Variables

Single variables can be transferred to the watch list. To do so, select the variable line and press the <Watch> button, or use the context menu. Alternatively, select the desired variables via double click.

Long variables which cannot be fully displayed may be transferred to the Long Variable Display on the buffer display tabs. Select the variable with the right mouse button and the context menu item <full value size>.

6.1.3. Changing Variable Values

You may change the variable values with types 'Var' and 'Intern' at any time. Use the item <edit value> in the context menu.

6.2. Watch List

The Watch List contains a subset of the Variables List. Variables are transferred to the watch list:

- From the Variables List as described in chapter Selecting Variables on page 27
- or
- From the source display via double click & drag

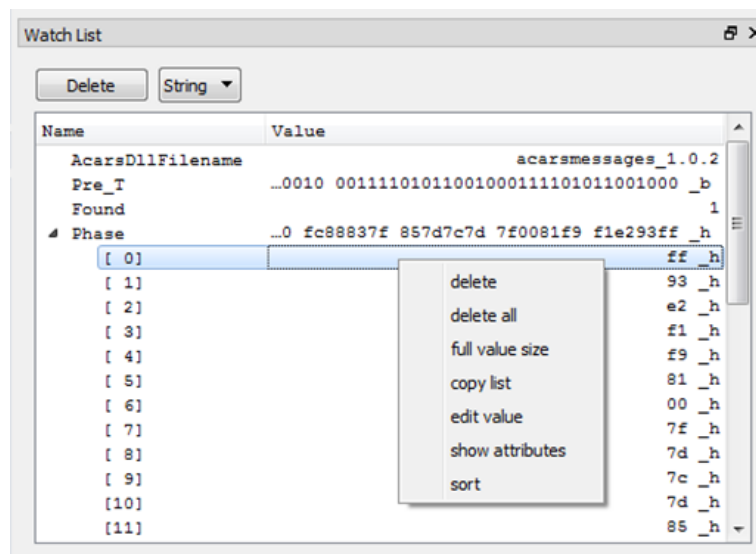


Figure 25: Watch List

These variables value annotation formats can be selected individually for each variable:

Selection	Description
Hex	hexadecimal
Dec	decimal (first 32 bits)
Oct	octal
Bin	binary
String	ASCII-String

Table 16: Annotation Settings

Array variables can be expanded to see each element separately. The list can be further managed via the right mouse button context menu:

Function	Description
delete	delete selected variable
delete all	clear the watch list
full value size	copy the selected variable to the Long Variable Display
Copy list	copy the list to the clipboard
edit value	modify the selected variables value. The values can be mediteddirectlyas well.
show attributes	show the variables size and array dimensions
sort	toggle between automatic and manual sorting methods. Automatic sorting will sort the table based on a selectable column (name or value). Manual sorting allows placing a new variable via drag & dropping at a dedicated position. The table can be sorted by later internal drag & drop operations as well.

Table 17: Context Menu Functions

6.3. Long Variable Display

This display is provided to show the value of long variables to their full extent. Variables are transferred to the long variable list:

- From the Variables List as described in chapter Selecting Variables on page 27 or
- From the Watch List in the same way or
- From the source display by double clicking & dragging it to the display's tab label

As a result, the tab label will assume the variable's name and the value will be shown in the selected format. Possible display formats are binary, hexadecimal and ASCII text. The box can be edited and the values will be updated on activation of <Apply>.

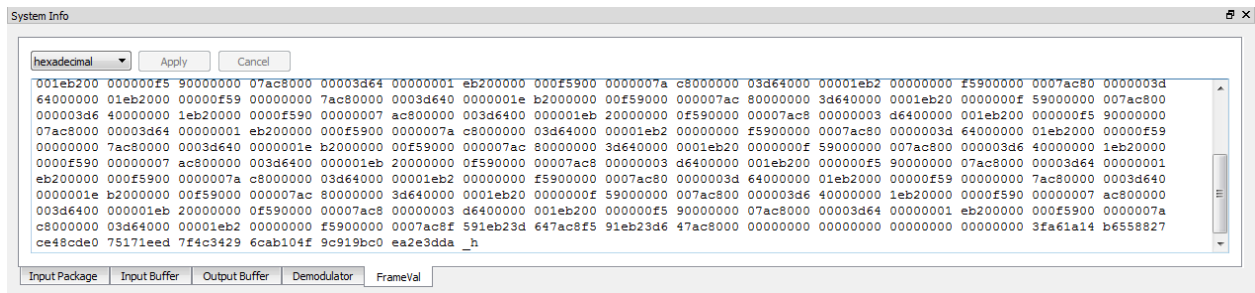


Figure 26: Long Variable Display

7. Displays

7.1. Buffers

Incoming and outgoing data pass through intermediate buffers as displayed in Figure 26. The input package contains the demodulated bit stream received from the demodulator. This package is transferred to the input buffer after optional pre-processing. The main program receives all inputs (e.g. for commands *Search...* and *Get...*) from the input buffer. Command *Out...* will write to the output buffer which is flushed before the next input package is received.

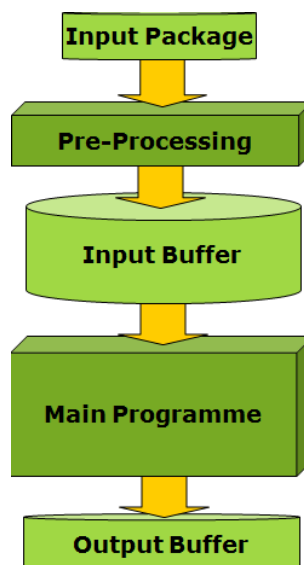


Figure 27: Buffering Concept

7.1.1. Input Package

The input package bit stream is shown in conjunction with a set of parameters received in parallel. The bit stream can be displayed in most or least significant bit first order, controlled via the checkbox <LSB first>. The display also shows intermediate results of pre-processing steps, which can modify the package in question. The additional parameters on the right of the display are as follows.

Phase	<input type="text" value="Production"/>
Package Size	<input type="text" value="45"/> [Symbols / Chan]
No. of Channels	<input type="text" value="1"/>
Symbol Size	<input type="text" value="1"/> [Bits / Symb]
Symbol Rate	<input type="text" value="1800.0"/> [Baud]
Time	<input type="text" value="14:10:35.344"/>
Date	<input type="text" value="2016-04-25"/>

Figure 28: Input Package Display Parameters

7.1.1.1. Phase

This parameter indicates whether the package is received during search or production phase. The search phase is the initial phase of automatic production process, when all possible modems are tested. This phase only can be observed in <Decoder Debugger Online> mode. The production phase follows the search phase if a positively responding modem has been found. In <Decoder Debugger> mode, only the production phase is simulated.

7.1.1.2. Package Size

This value describes the current number of input symbols per input channel contained in this input package. To obtain the total number of received bits, this value must be multiplied with the symbol size and the number of channels.

7.1.1.3. No. of Channels

Number of demodulator channels which has been defined as a demodulator parameter for the respective modem.

7.1.1.4. Symbol Size

This indicates the number of bits per incoming input symbol. This value corresponds to the modulation order (or "valence") defined with the demodulator.

7.1.1.5. Symbol Rate

This value is the result of the actual symbol rate measured. It may deviate from the nominal symbol rate defined with the demodulator.

7.1.1.6. Time/Date

This time designates the arrival time of the first symbol of this package. It can be referred to the system clock or file time, depending on the settings in DANA.

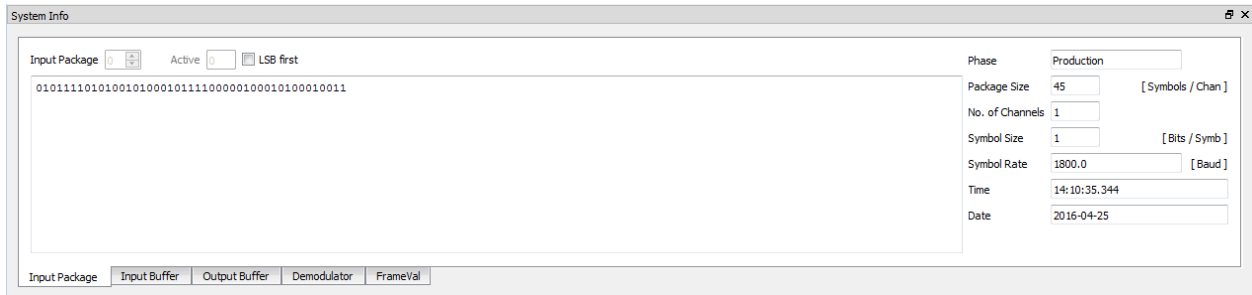


Figure 29: Input Package Display

Multiple input packages will be present when the decoder executes commands to split a multi-channel input package into single channel packages. In this case, the spin box <Input Package> is enabled to switch the display between channels. The package currently treated by pre-processing is indicated in the box <Active>.

7.1.2. Input Buffer

The input buffer is displayed in most significant bit first or least significant bit first order. The input buffer pane additionally shows the current position of the read pointer after "=>" respectively before "<=", and as an absolute value in the box <Read Position>. The number of bits from the read pointer to the last incoming bits is displayed in the box <Remaining Bits>. The displayed number of bits before the read pointer can be set in the spin box <History Length>. <Read Position> is the absolute value of the read pointer position, designating the number of bits since the start of input bit stream.

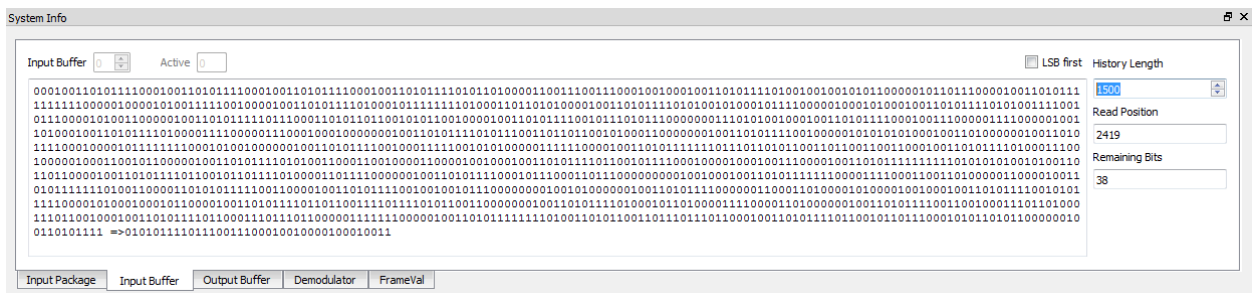


Figure 30: Input Buffer Display

The characters „<B” in the input buffer pane marks the beginning of a detected burst, “” designates a burst end. Input symbols exceeding 1 bit are separated by blanks.

7.1.3. Output Buffer

The output buffer display shows the result of command <Out...> in unfiltered XML format. It is flushed at the time of data exchange, i.e. the time a new input package is requested to continue decoder processing.

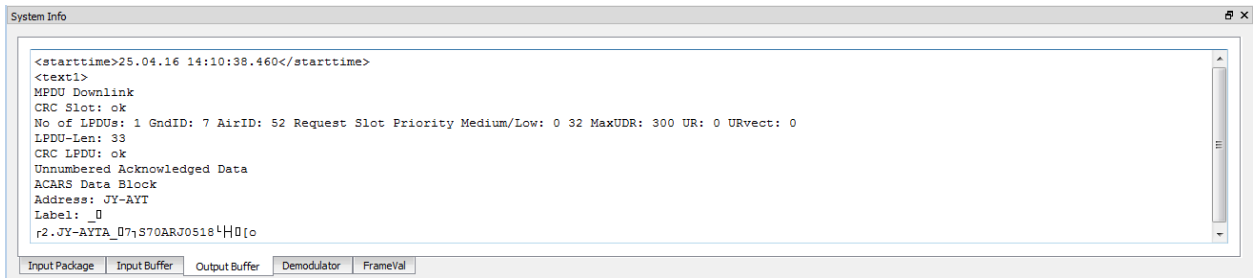


Figure 31: Output Buffer Display

7.2. Demodulator Parameters

The tab <Demodulator> shows the current demodulator settings as they would be read by the decoder at this point. Parameters which are not relevant for the current type of demodulator are shaded in grey. They are displayed nevertheless as they may be relevant when changing the type of demodulator during the decoding program.

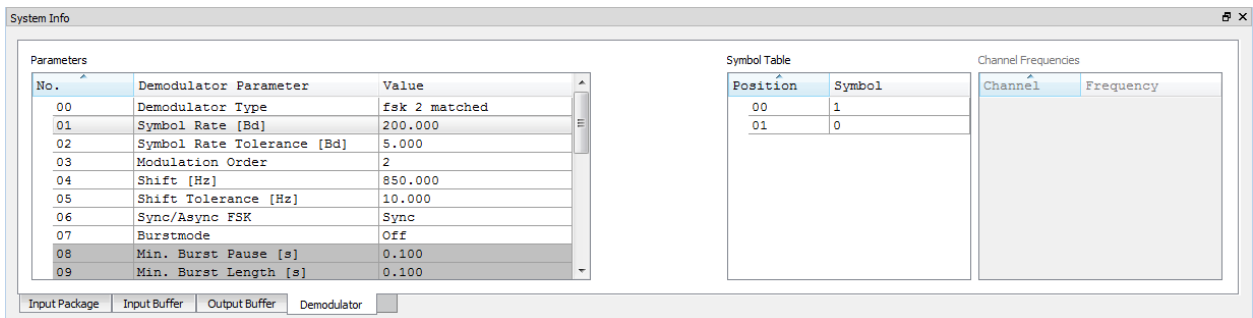



Figure 32: Demodulator Parameters Display

7.3. Result

This pane shows the decoded text results once the output buffer has been flushed. It will be present exclusively in Offline Mode as in Online Mode the software will use the SDA result fields.

The results can be displayed as complete raw XML text or previously pass a definable output filter. Call the filter table by

- <View><XML Filter>, <Ctrl> + <F>, or press 

A list of standard XML tags is shown which can be modified as required. Add or delete tags, and interpret the list via the selection in the top left drop-down list box.

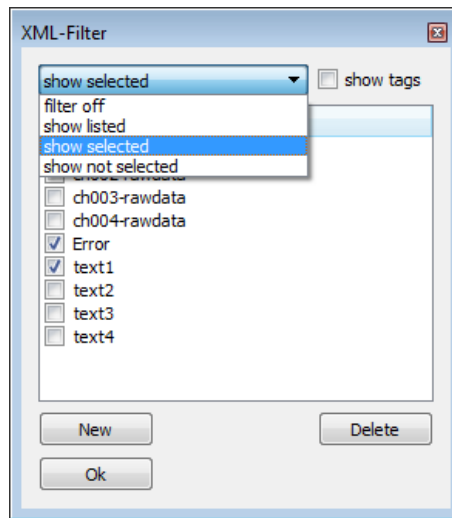



Figure 33: XML Filter Table

The content of a result file can be deleted at any time by

- <Test><Clear Result>, <F8> or press .

8. Decoder Programming Examples

The following simple example gives an initial impression of how to use the language commands and the program structure. The intended decoding function is to identify two different start patterns in one signal and to display the resulting ASCII texts.

```
*****
;*           Simple Example of Decoder Programming           *
;* Following the occurrence of pattern 0011 1100 and 0011 1110 *
;* (each repeated twice) two messages of length 208 bit will be *
;* decoded as ASCII characters                               *
*****
VARDEF
; Variables and initialisations
; (Standard variable size is 32 bit)
Found
Tolerance = 0
Repeat = 2
GapLimit = 500 ;

256: Frame; Input variable of size 256 bit

ENDVARDEF
*****
START

;Main program
NewSync:

Frame = 0; Clear input text field

; Search for the twice repeated pattern within the next 500 bit
SearchPattern(0011 1100_m, Repeat, Tolerance,, GapLimit,,Found)
;change into 0011 1110_m for second message
;      or 0011 11X0_m for both messages

If(Found) ; If search successful
    ;Read initial pattern and message (208+16 bit) from input buffer
    GetFrame(224,Frame)
    ;Initial pattern will be shifted out (mentiont LSB-first-logic)
    Frame = Frame >> 16
    Ident(); Message "Modem detected"
    OutText(Frame,1) ;Output of ACII-Text on output channel 1
Else ; If search failed
    Fail() ; Message "Modem not detected"
EndIf

GoTo(NewSync); Jump back to program start

END
```

Figure 34: Decoder Programming

The following files are available for this purpose in the directory:

- Source code: "Example1e.txt" (/examples/ddl)
- Compiled code: "Example1e.bin" (/examples/ddl)
- Suitable signal: "Example1.wav" (/examples/ddl)

To test the program, proceed as follows:

- Create a new modem list
- Create a modem (named “Test”) in this modem list using the settings shown in Figure 35

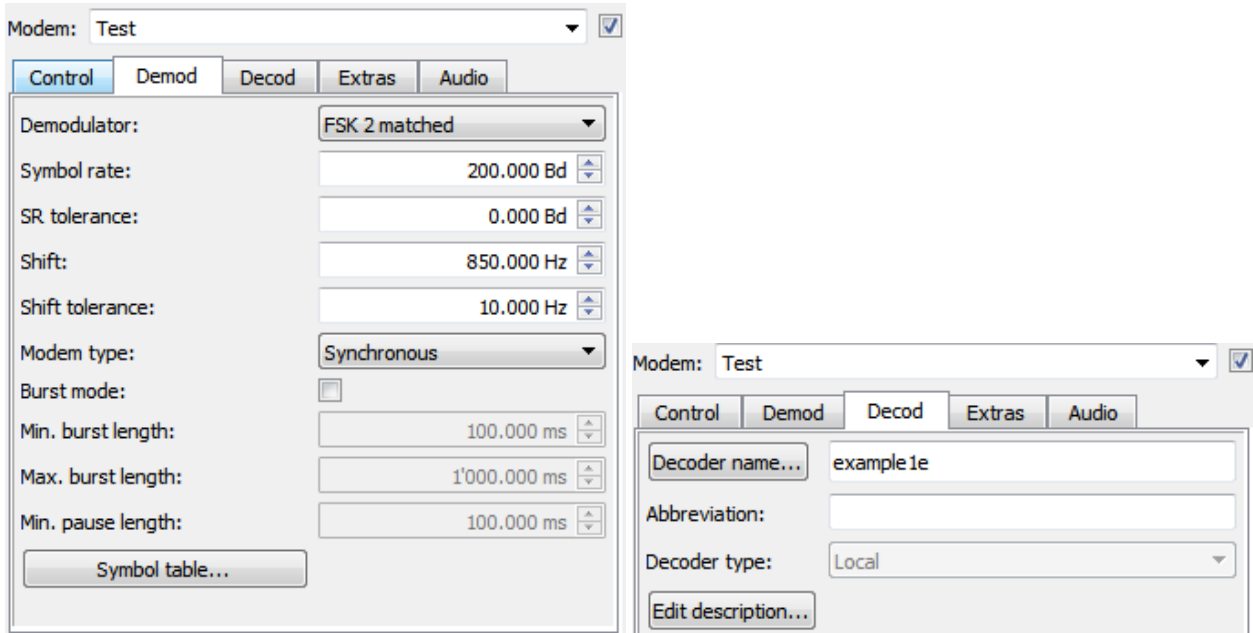


Figure 35: Demodulator/Decoder Settings

- Start DANA with a center frequency setting of 1,800 Hz and the source *File*
- Select and start the file “Example1.wav” in DANA
- Start automatic production (*Automat*)

The result display in Figure 36 shows a positive message of the production automat together with the decoded text.

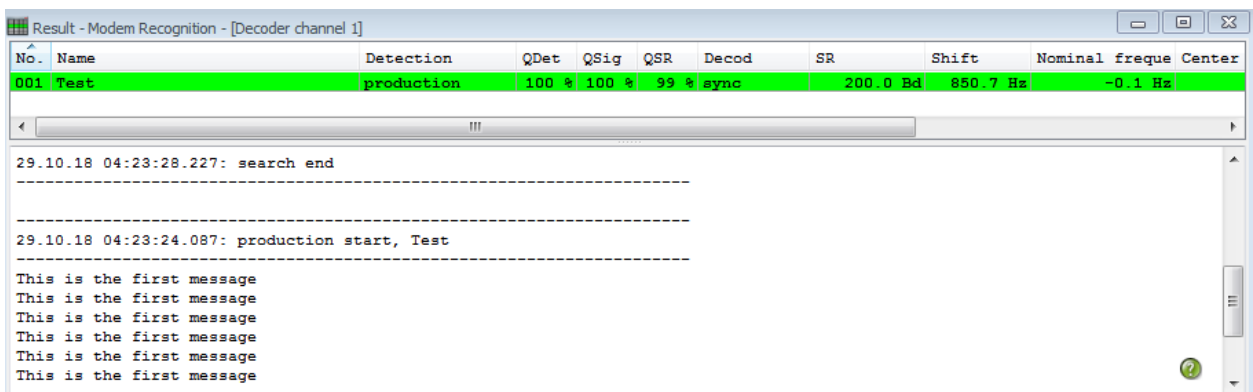


Figure 36: Result Display showing first message

- Change the search pattern as indicated in the comment lines in the section SearchPattern
- Compile the source code and check for compiler error messages
- Restart automatic production (*Automat*). The second message is displayed.

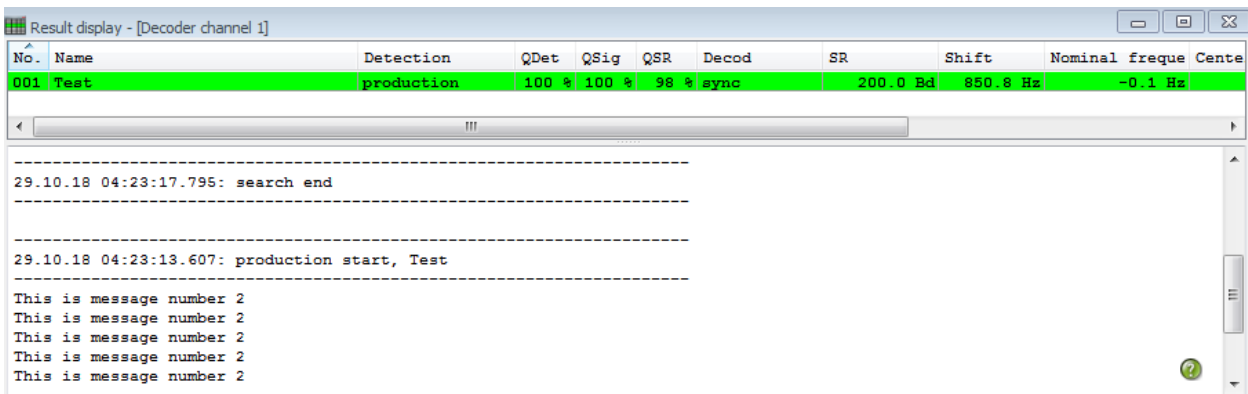


Figure 37: Result Display showing second message

As stated in the comments it is also possible to identify both patterns using X-wildcard characters and to output both messages successively.

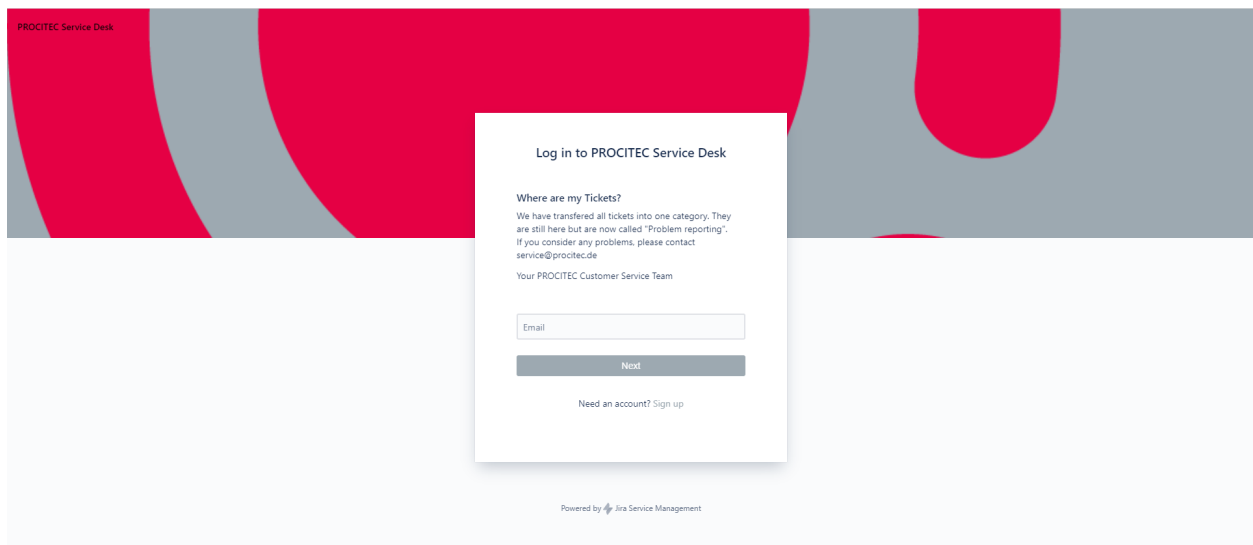
A. Support

Requests and suggestions?

All requests or suggestions regarding our go2signals product-range are very much appreciated; we would be delighted to hear from you.

Any questions? We are happy to assist you!

If you have any further questions, please do not hesitate to contact our Support Team for rapid assistance – just raise a service request at: <http://servicedesk.procitec.com>.



PROCITEC GmbH
Rastatter Straße 41
D-75179 Pforzheim
Phone: +49 7231 15561 0
Web: www.procitec.com
Email: service@procitec.com

List of Figures

1.	Overview of Decoder Creation	2
2.	Load Decoder Editor	3
3.	Decoder Editor	4
4.	Menu Bar	5
5.	File Menu	5
6.	Edit Menu	6
7.	Extras Menu	7
8.	Help Menu	7
9.	Toolbar	8
10.	Automatic Command Completion	9
11.	Decoder Status Bar	10
12.	Example of a Decoder Program	11
13.	Decoder Data Flow	12
14.	Result Display on Calling the Command Ident	15
15.	Message on Successful Compilation	16
16.	Display of Runtime Error	16
17.	Decoder Debugger – Offline Operating Environment	17
18.	Decoder Debugger – Offline Main Window	18
19.	Starting Demodulator Output Records	19
20.	Online Debugging Concept	20
21.	Select Decoder Debugger	21
22.	Decoder Debugger - Online Main Window	21
23.	Loaded Modems List	23
24.	Variables List	26
25.	Watch List	27
26.	Long Variable Display	29
27.	Buffering Concept	30
28.	Input Package Display Parameters	31
29.	Input Package Display	32
30.	Input Buffer Display	32
31.	Output Buffer Display	33
32.	Demodulator Parameters Display	33
33.	XML Filter Table	34
34.	Decoder Programming	36
35.	<i>Demodulator/Decoder Settings</i>	37
36.	Result Display showing first message	37
37.	Result Display showing second message	38

List of Tables

- 1. Decoder Editor Color Assignments 3
- 2. Decoder Editor Menu Items 5
- 3. Decoder Editor File Menu Items 5
- 4. Decoder Editor Edit Menu Items 7
- 5. Decoder Editor Extras Menu Items 7
- 6. Decoder Editor Toolbar Icons 8
- 7. Decoder Editor Shortcuts 9
- 8. DDL Search Commands 12
- 9. DDL Read Commands 13
- 10. DDL Bit Manipulation Commands 13
- 11. DDL Check and Correction Commands 13
- 12. DDL Output Commands 14
- 13. DDL Output Commands 14
- 14. DDL Pre-Processing Commands 15
- 15. Variables Attributes 26
- 16. Annotation Settings 28
- 17. Context Menu Functions 28