User manual V2.1.01

WAVECOM W61BV BitView

by WAVECOM ELEKTRONIK AG

PUBLISHED BY
WAVECOM ELEKTRONIK AG
Hammerstrasse 8
CH-8180 Buelach
Switzerland

Phone +41-44-872 70 60
Fax +41-44-872 70 66
Email: info@wavecom.ch
Internet: http://www.wavecom.ch

Printed: Tuesday, April 15, 2008, 14:10:01

Contents

Introduction 1

General 1
Revisions 1
Open Issues 1
Requirements 1
Limitations 1

Installation 2

Setup 2
Paths 3

Getting Started 3

Program Start 3

Menu 5

Bit Stream Processing 5
Analysis Sets 6
Reports 7
Properties Window 8
History Explorer Window 9
Toolbox 9
Preferences 9
Layout Settings 10

Function Library 14

Source/Sink 14
Import Text Data 14
Import Hex Data 14
Import Binary Data 14
Import IAS Bitstream 14
Export Text Data 15
Synchronization 15
Preamble 15
Binary Modulation 15
NRZ-I 15
NRZ-M 15
NRZ-S 16
Bi-Phase-L (Manchester) 16
Bi-Phase-M 16
Bi-Phase-S 17
DBi-Phase-M 17
DBi-Phase-S 17
Bit Manipulation 18
De-Stuffing (HDLC) 18
Mirroring 18
Rotation 19
Shift 19
Polarity 19
De-Interleave Bit Block 20
De-Interleaving Stream 20

AND / OR / XOR / NOT 21
Extraction 21
Cutting 22
Decoding/Equalizer 22
Viterbi-Decoding 22
De-Puncturing 23
Difference-Decoding 23
BCH-Decoding 24
CRC & Polynomial 24
CRC (1..32) 24
CRC-8 25
CRC-10 25
CRC-12 26
CRC-16 26
CRC-CCITT 26
CRC-32 27
Channel Decoding (Protocol) 27
ARQ-E 27
SITOR 28
FEC-A 28
BAUER 28
HNG-FEC 28
RUM-FEC 29
ITA-3 (M.342) 29
ITA-5 29
PSK-31 30
Source Decoding (Alphabet) 30
Latin 30
Third-shift Greek 30
Cyrillic 30
Tass-Cyrillic 31
Third-shift Cyrillic 31
Hebrew 31
Arabic Baghdad-70 31
Arabic Baghdad-80 (ATU-80) 31
Bulgarian 32
Swedish 32
Danish-Norwegian 32
German 32
French 32
US 32
ASCII 33
UNICODE 33
UTF-7 33
UTF-8 33
Analysis Tools 33
Bit Statistic 33
Autocorrelation 34
Signal Duration 34
Bit Sync Analysis 35

Custom Library **38**

Want to Roll Your Own functions? 38
Adding a Custom Function 40
Constraints 41
Important Notes 41
Steps to Write a Custom Function in C# .NET 41
Steps to Write a Custom Function with Matlab 42
Source Code Template / Example (C# .NET) 43
CustomLibFunction.cs 43
Source Code Template / Example (C# .NET for Matlab) 46
BVCustLibMatlab.cs 46
Source Code Template / Example (Matlab) 53

MatlabFunction.m 53

Glossary of Terms **55**

Index **57**

Introduction

General

BitViewTool enables the user to analyze any bit stream. The range of functions extends from the display of a bit stream in various formats, simple bit stream manipulations, over statistical functions to complex mathematical functions and functions based on coding theory. The tools are targeted at users with experience in bit stream analysis. To understand some of the functions a comprehensive, mathematical knowledge is a prerequisite.

BitViewTool supports the Windows 2000, XP, 2003 Server and Vista operating systems.

Revisions

Version	Date	Changes
1.1	10.May.2007	
2.0	05.Nov.2007	Installation folders changed 'Hide on close' preference added Layout settings removed from context menu Layout settings now in the property grid Graphic layout added 'Inversion' function name inversion changed to 'polarity' Bit Sync Analysis added Custom library updated
2.1	14.Mar.2008	Hexadecimal view added Enhanced printer dialog Matlab custom libraries
2.1.01	15.Apr.2008	General overwork : ➢ Improved readability ➢ Extended explanations 'Parameters' window changed to 'Properties' window

Open Issues

Version	Date	Open Issue
2.1	14.Mar.2008	Default import will be HF IAS bitstream Default port for License check is fixed

Requirements

.NET Framework version 2.0 must be installed. The framework is included in the setup and is installed if missing on the system.

An installed W61PC card with BitView enabled.

Limitations

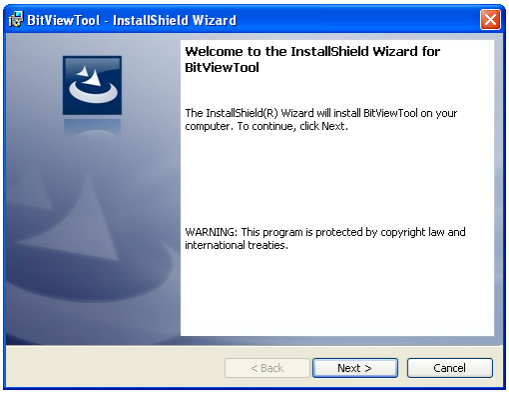
At present, the maximum number of bits that can be imported is limited to 500,000.

Bear in mind that some formatting functions such as bit highlighting are consuming a lot of CPU power and may require considerable time to complete, especially on slower machines. Reducing the number of imported bits will speed up the application.

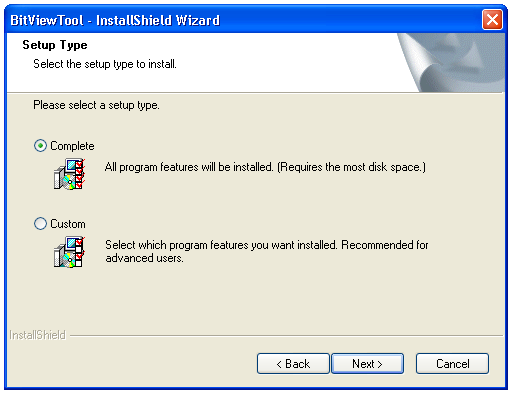
Installation

Setup

To install the application, click **Setup.exe.** Files are then unpacked and copied to the installation folder. Ini files are not generated.

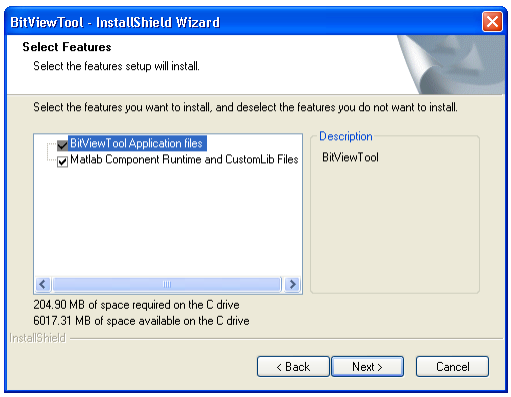


Two types of installation are available, **Complete** and **Custom.**

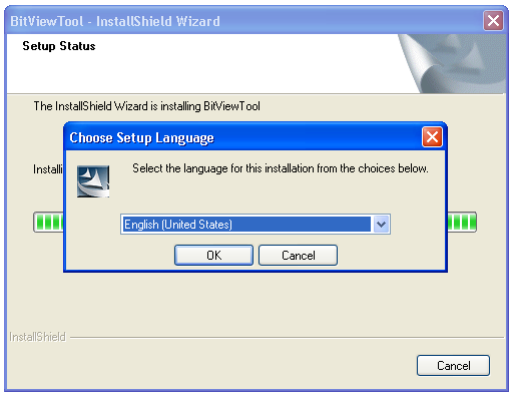


Complete installation will install BitViewTool, the Matlab runtime and the Matlab CustomLib examples on your system.

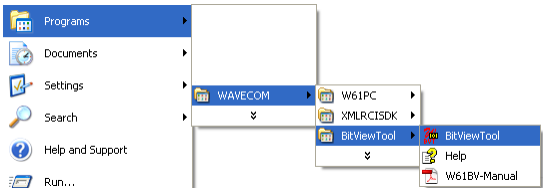
Custom installation will only install the selected components on your system.



Depending on your selection, a language dialog will appear during installation of the Matlab runtime.



By default BitViewTool will be installed in the WAVECOM folder, where other WAVECOM products may be installed.



BitViewTool may be uninstalled by using the **Add/Remove Programs** item found in the **Control Panel** menu.

Paths

Examples and CustomLib files are copied to the following folder:

Windows XP and earlier:

- Documents and Settings\All Users\Documents \WAVECOM\BitViewTool\

or

- Documents and Settings\All Users\Shared Documents \WAVECOM\BitViewTool\

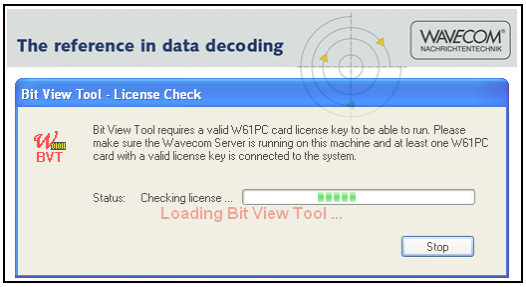
Windows Vista:

- Users\Public\Public Documents\WAVECOM\BitViewTool\

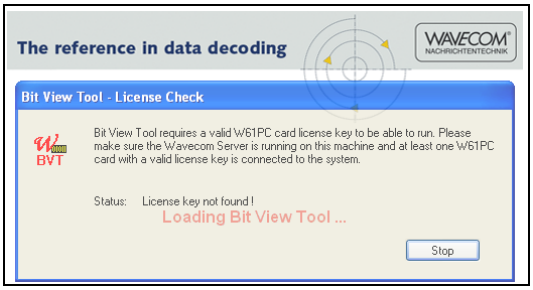
Getting Started

Program Start

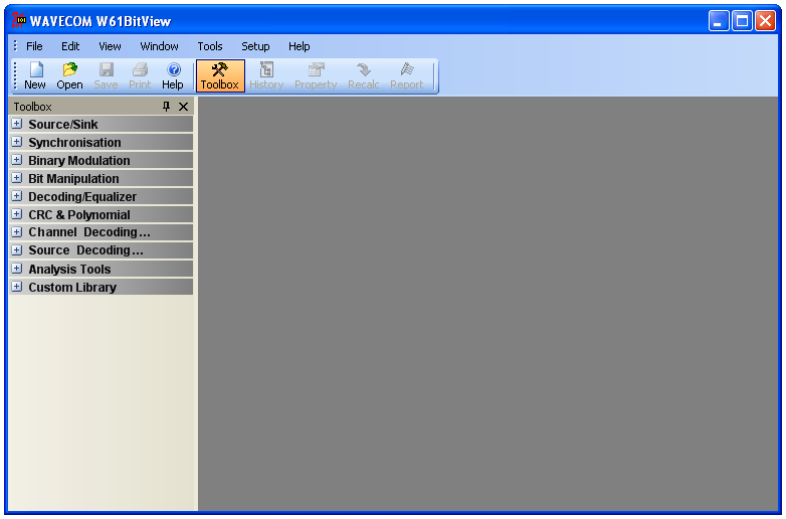
Starting the program will introduce a license check procedure.



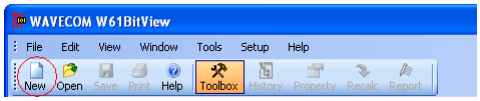
If no valid license key was found, the following message appears on the screen.



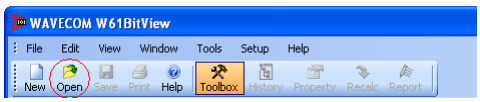
If a valid license key was found the application is started. The **Toolbox**, which contains the function library, is displayed and enables the user to import a bit stream from a selection of different sources.



Alternatively, using the **New** button from the **Toolbar**, an empty document window is opened which allows the user to manually create a bit stream, or copy and paste a bit stream from another source.

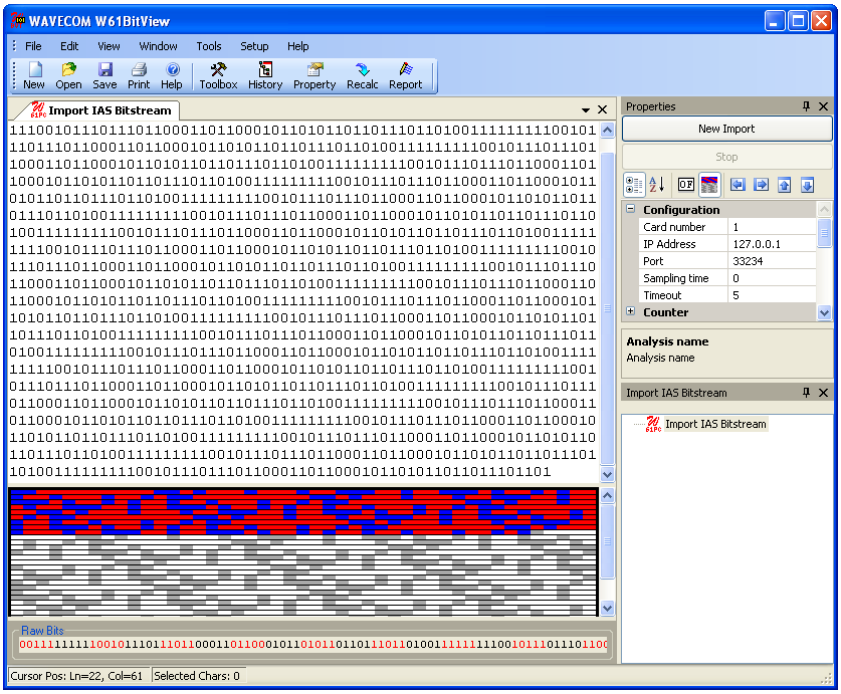


Another option, using the **Open** button from the **Toolbar,** allows a previous saved **Analysis Set** (stored in a .XML file) to be opened (see later in this manual for details).



Bit Stream Import

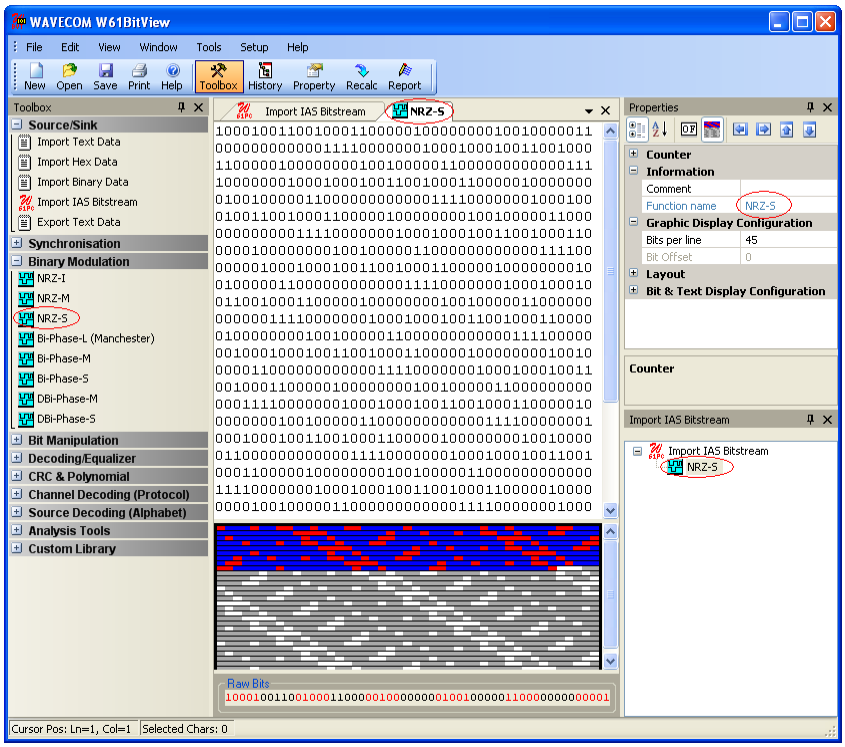
An imported bit stream is shown in a document window, and **Properties** and **History Explorer** windows are opened. In general, the **Parameters** window displays all the properties of a selected function, and the **History Explorer** window shows the dependencies of all functions in a tree view.



Menu

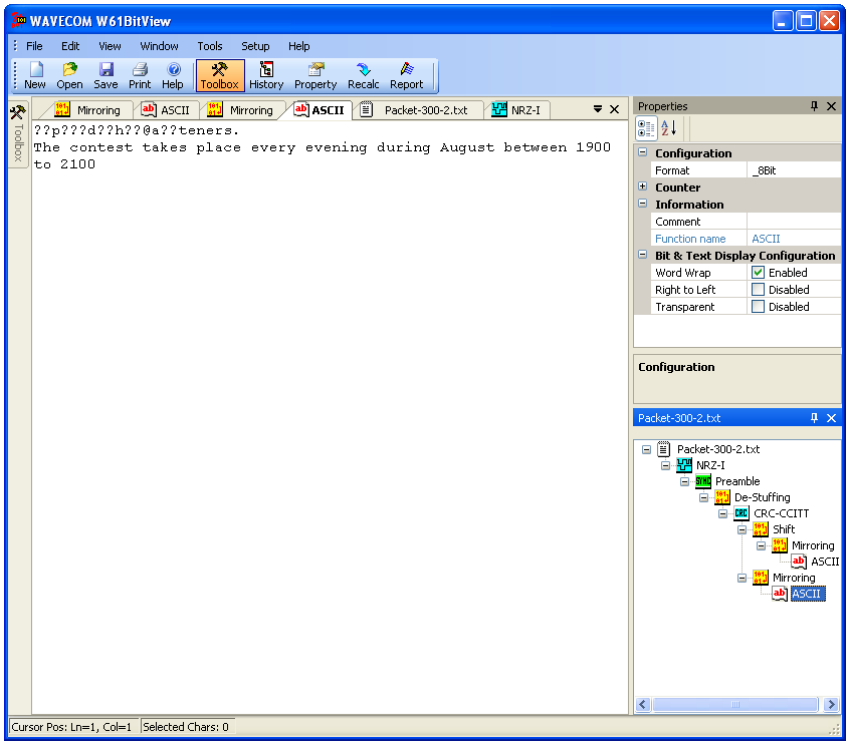
Bit Stream Processing

An imported bit stream may be processed using any of the functions found in the library. The processed bit stream is shown in a new document window. All document windows are shown as tabbed windows.



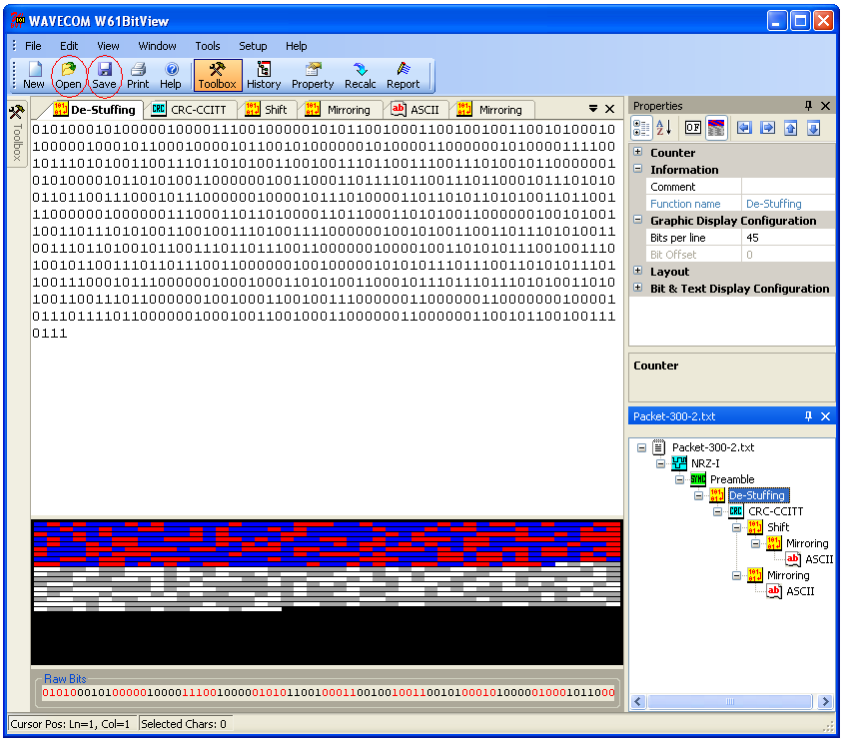
Functions may be added to form a so-called **Analysis Set**, which contains an imported bit stream and the configured functions operating on the bit stream. The user may define and create different analysis paths,

as may be seen in the History Explorer. The imported bit stream is processed according to the configuration settings of the selected functions.



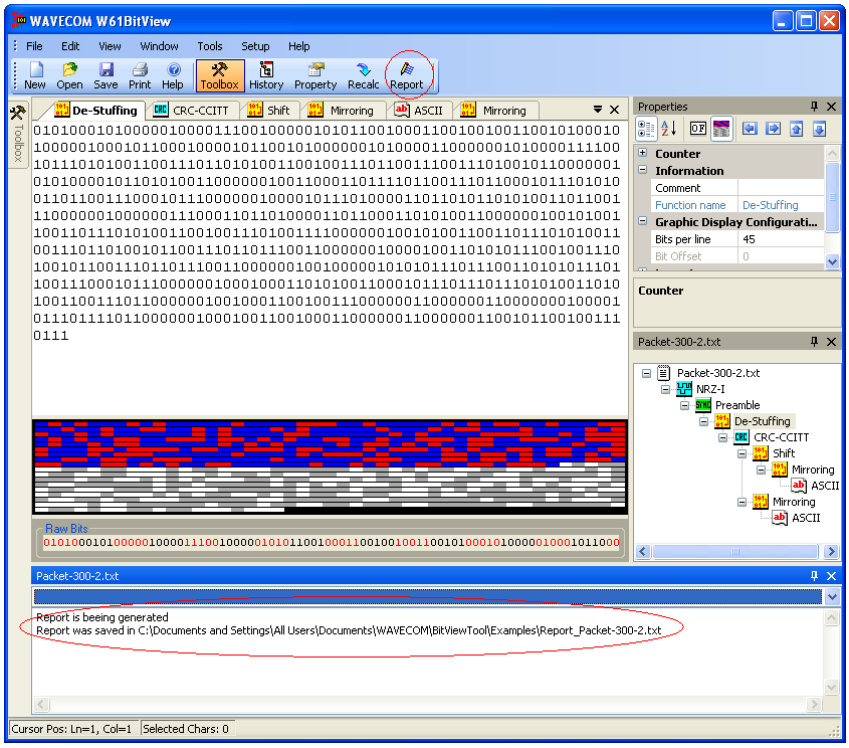
Analysis Sets

Using the **Save** button in the toolbar, **Analysis Sets** may be saved in an XML file, i.e. the imported bit stream and the parameter settings of the selected functions are stored. Using the **Open** button, an Analysis Set may be reloaded at any time.

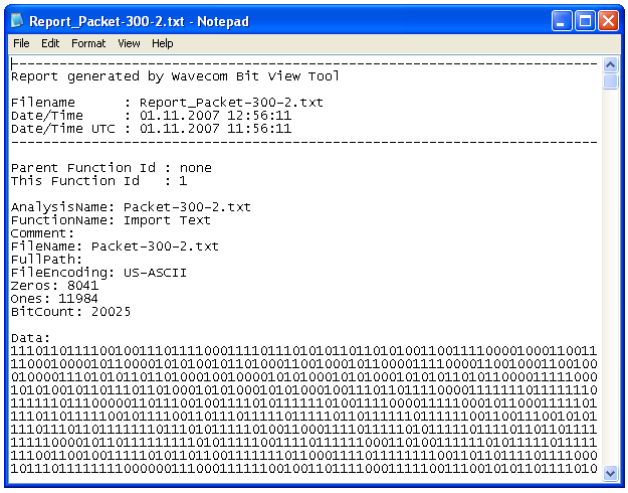


Reports

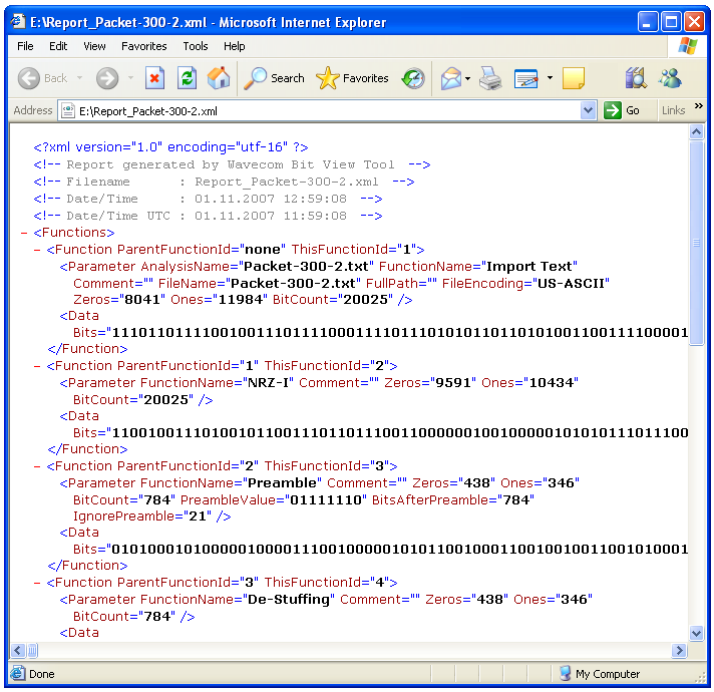
Using the **Report** button a complete Analysis Set may generated and saved as a **text** file or a **XML** file.



Example of a report stored in a **text** file.



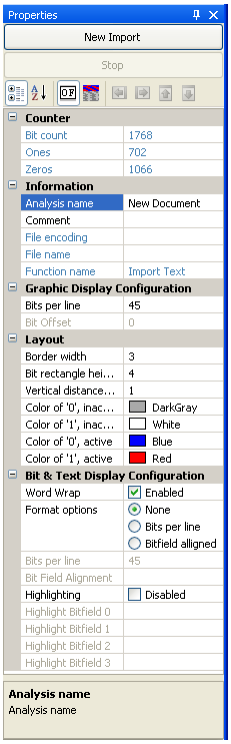
Example of a Report stored in a **XML** file.



Properties Window

The parameters in the **Properties** window are grouped into different categories providing the operator with information about actual parameter settings and - more important - allowing the operator to configure each function and to add comments.

Detailed information about the selected parameter is displayed in the Help window below the Properties windows.



As from release 2.0 with the introduction of the graphic display, three new categories have been added to the Parameters Window.

1. Graphic Display Configuration
2. Layout
3. Bit & Text Display Configuration

Please refer to the Layout Settings section to read more about these settings.

History Explorer Window

The **History Explorer** window provides a quick overview of the actual analyzing process. It allows the operator to try out different function paths with different parameter settings and enables instant comparison of the results of these trials.

Functions may be re-arranged and deleted using the mouse pointer (drag and drop) in combination with the modifier buttons (CTRL, ALT, SHIFT) and the right-click menu.

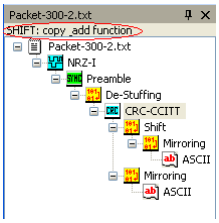
No button pressed : Move dragged function plus all sub-functions

CTRL pressed : Copy and insert dragged function

ALT pressed : Copy and add dragged function plus all sub-functions

SHIFT pressed : Copy and add dragged function

Detailed information is displayed on top of the window, when a button is pressed while dragging.



Toolbox

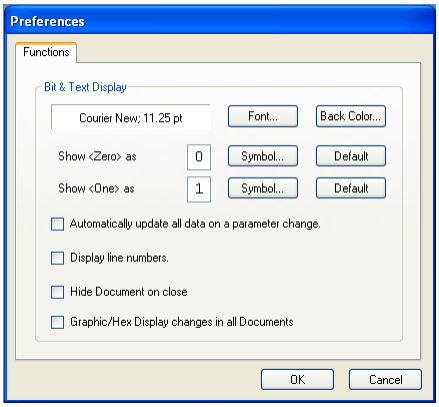
The **Toolbox** menu is divided into libraries, and each library contains one or more functions.

Note:
The **Custom Library** is not visible unless a custom function has been added.
The **Analysis Tools** are not added to the History Explorer tree and are not persistently stored.

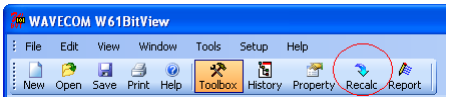
Preferences

The **Preferences** dialog box may be selected from the **Setup** menu.

Substitution symbols for logical zero and logical one may be directly edited or selected using the appropriate **Symbol** button.



If **Automatically update all data on a parameter change** is ticked, all functions are automatically re-calculated when the operator changes a parameter. Uncheck the tick box if this behavior is not desirable, and use the **Recalc** button in case a recalculation is necessary.



If **Hide document on close** is ticked, a document is hidden when closed, but remains in the History Explorer. Clicking the function associated with the document in the History Explorer will make the document visible again.

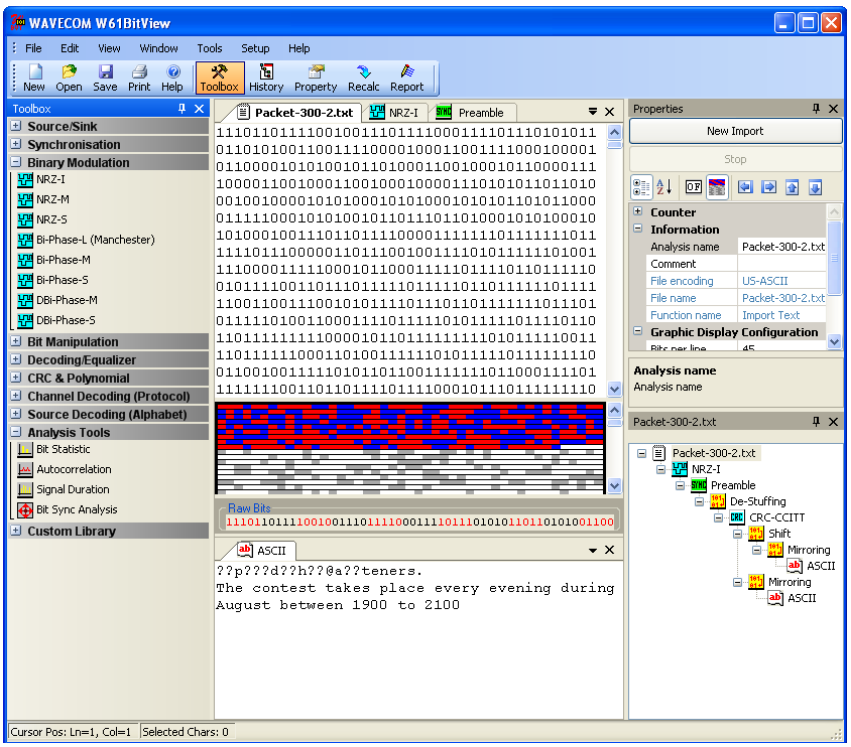
To remove the function completely, select the appropriate function in the History Explorer and press the **Delete** key on your keyboard.

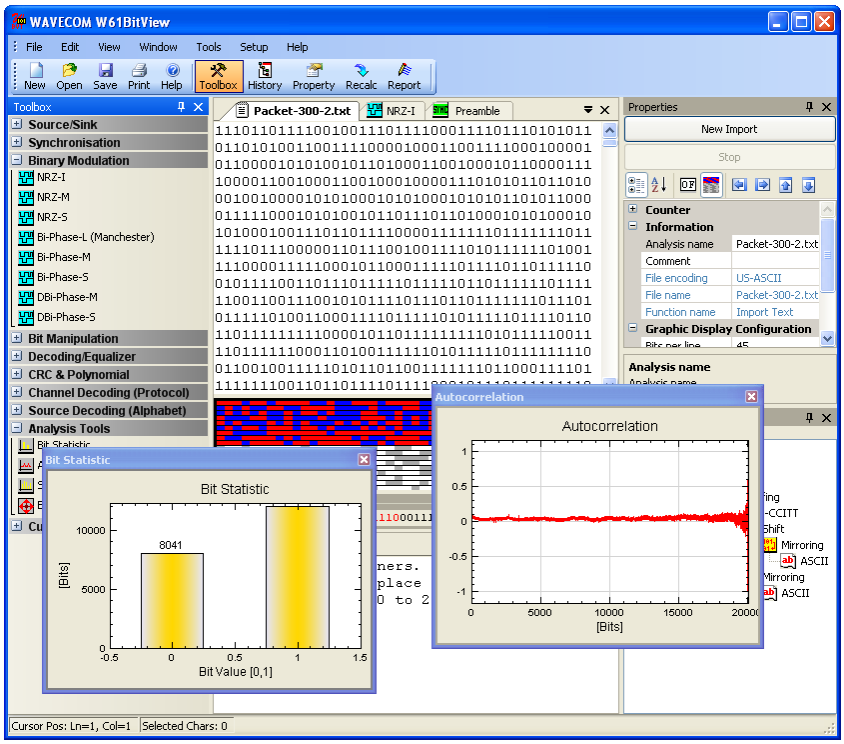
If this option is not checked, documents are completely removed when closed. Closing the root document will close and remove all other functions and their associated documents .

If **Graphic/Hex display changes in all documents** is option selected, all the documents will have the same display settings, i.e. if the display is changed from Graphic to Hex in one document, all other documents will change their display type as well.

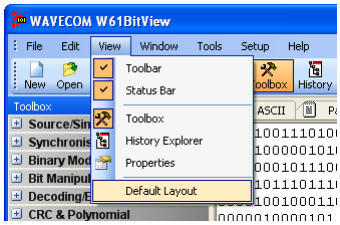
Layout Settings

At run time, the user can **freely drag and drop** all windows to re-arrange them according to the preferred layout. In addition the Parameters, History Explorer and Toolbox windows use **auto-hide** functionality.

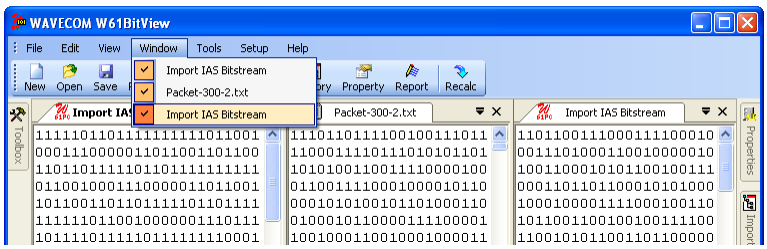




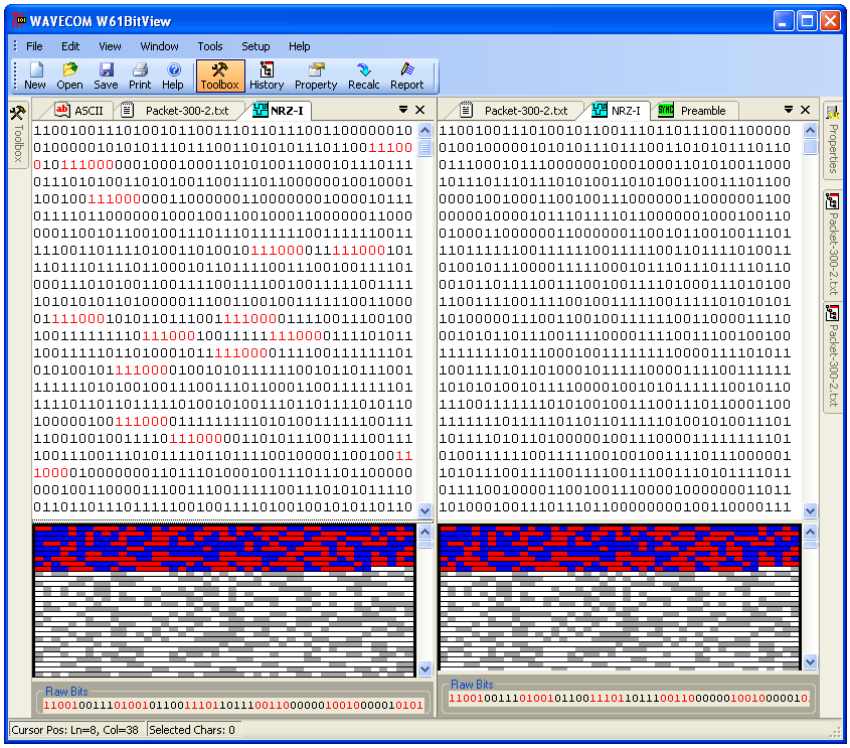
Use the menu entry **View > Default Layout** to use the default layout of the windows.



More than one Analysis Set may be active at a time. Using the **Window** menu allows the user to show or hide these Analysis Sets.



Using the **Bit & Text Display Configuration** category in the Properties window enables the operator to use different display format options.



Word Wrap

Checking the **Word Wrap Enabled** tickbox enables word wrapping in the bit and text document.

Bits per line

This parameter allows displaying a specific number of bits per line. Choose **the Bits per line** radio button under format options to enable this feature.

Bitfield alignment

Whenever the specified bit pattern is found in the bit stream, a new line is started, i.e. a line break is inserted. Choose the **Bitfield aligned** radio button under format options to enable this feature.

Highlighting

The bit stream is searched for a specific bit pattern and when found the pattern is marked. A maximum of four different search patterns are possible. Check the **Highlighting** tick box to enable this feature and enter the search patterns in the appropriate text fields.

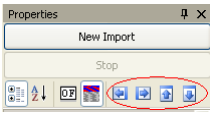
Graphic Display

A graphic display is associated with the bit stream and may be selcetd from the top of the Properties window. The size of the graphic display can be changed vertically by dragging its top border.

The **Layout** category in the Properties window lets you change the appearance of the graphic display.

In the **Graphic Display Configuration** category in the Properties window, the number of bits per line can be changed. This feature can be used to find periodic bit patterns in the bit stream by changing the number of bits per line until a repeating bit pattern is visible. It is much easier to find those patterns using the graphic display than to use the bit and text display.

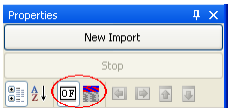
The arrow buttons on top of the Properties window are designed to move the active selection in the graphic display. The selected bits are displayed as **Raw Bits** in the bottom of the graphic display.

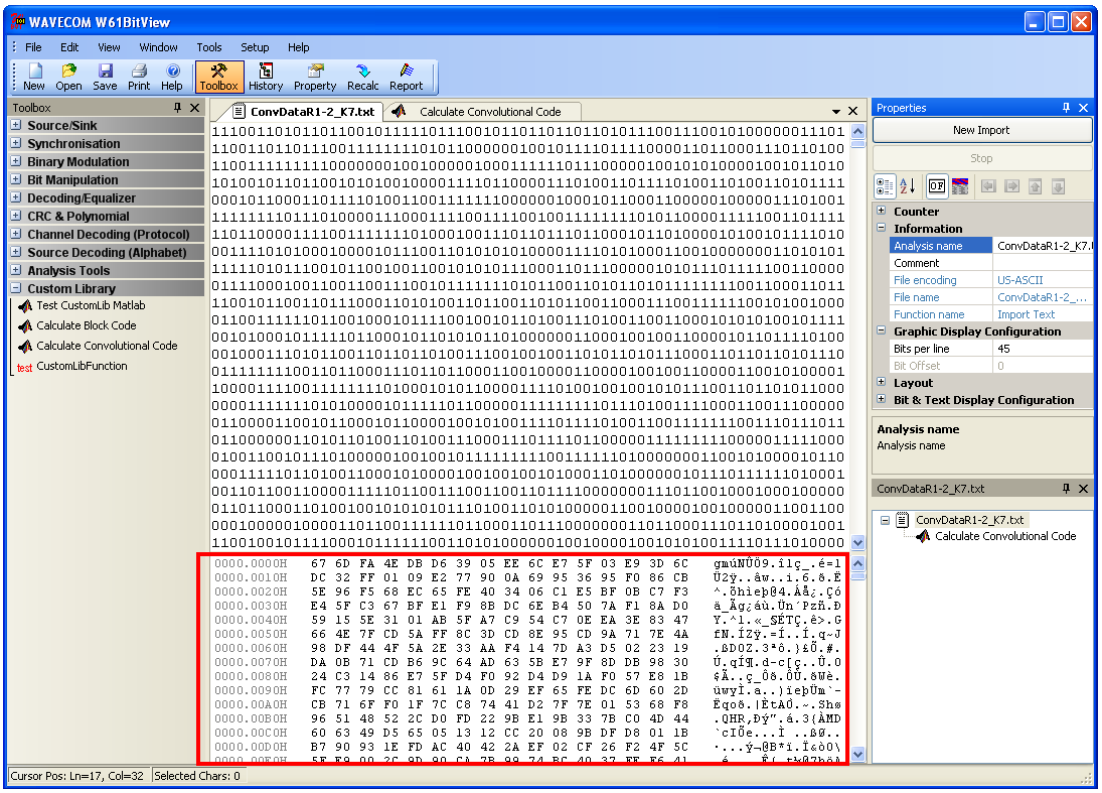


It is possible to zoom into the graphic display. Hold the left mouse button down and select the area that is to be expanded. A right click on the graphic display shows the context menu for un-zooming the view.

Hex Display

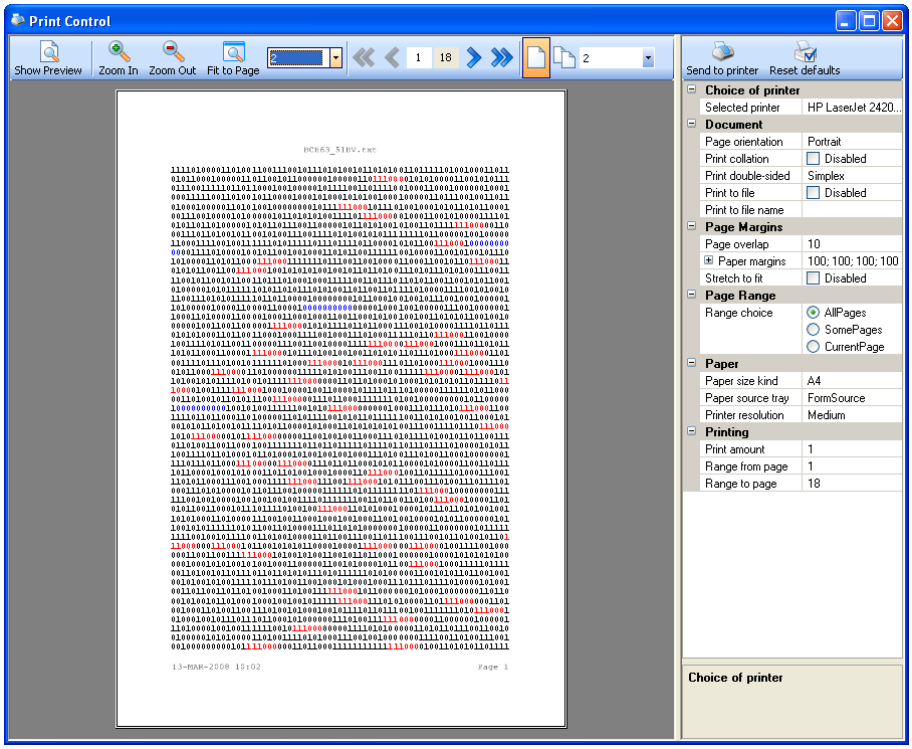
Another view of the bit stream is the hex display, selectable from the top of the Properties window.





Printer Dialog

The printer dialog is used for print preview and the printer settings. Use it by clicking the **Print** button or by selecting the menu entry **File > Print**. All layout settings, i.e. highlighting, alignment or bits per line, are supported.



Function Library

Source/Sink

Import Text Data

This function imports a bit stream from a text file. Only ASCII ones (0x31) and zeros (0x30) are considered as valid characters, others values are ignored and will not be loaded.

Example: “0110w700” is imported as “011000”.

Import Hex Data

This function imports a bit stream in hexadecimal form from a text file. Only ASCII figures from “0” to “9” and letters from “A” or ”a” to “F” or ”f” are considered as valid characters.

Example: “a1bg0c1kd0” is imported as “10100001101100001100000111010000”

Import Binary Data

This function imports binary files and displays their content as a stream of ones and zeros.

Import IAS Bitstream

In addition to loading a bit stream from a file, streams can also be directly transferred from WAVECOM server, the application that manages WAVECOM decoder cards. In order to be able to connect to the server, the following settings are required:

Parameter	Value
Card number	Selection of decoder on the system
IP address	IP address or MS computer name of the PC that hosts WAVECOM server

Port	Port number of Remote Control Interface (RCI) of WAVECOM server
Sampling time	Sampling time in seconds. '0' means infinite
Timeout	Maximum time to establish a connection to WAVECOM server. If a connection to the server could not be established within this period of time, the application cancels the connection procedure

After importing an IAS bit stream, the **Bit Sync Analysis** function automatically opens.

Export Text Data

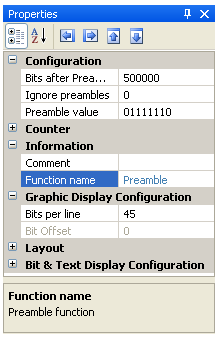
Writes the content of the current document into a text file. A **Save As** Dialog will appear to select filename and a folder for the exported file.

Synchronization

Preamble

In: Bit stream

Out: Bit stream



Function:

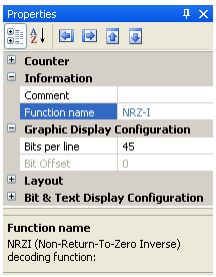
Searches for the **Preamble value** in the incoming bit stream and then writes the number of **Bits after Preamble** to the output. If the bit stream contains more than one preamble, the parameter **Ignore preambles** can be set for the function to skip a certain number of preambles.

Binary Modulation

NRZ-I

In: Bit stream

Out: Bit stream



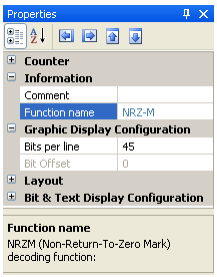
Function:

Changes the bit stream according to the "Non Return to Zero Inverse" (NRZ-I) decoding scheme, where no bit change represents a '1' and a bit change represents a '0'.

NRZ-M

In: Bit stream

Out: Bit stream



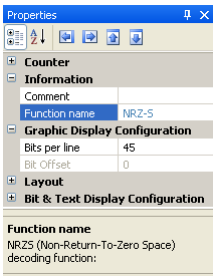
Function:

Changes the bit stream according to the "Non Return to Zero Mark" (NRZ-M) decoding scheme, where a bit change represents a '1' and no bit change represents a '0'.

NRZ-S

In: Bit stream

Out: Bit stream



Function:

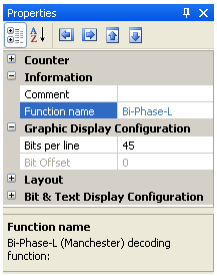
Changes the bit stream according to the "Non Return To Zero Space" (NRZ-S) decoding scheme, where no bit change represents a '1' and a bit change represents a '0'.

Note: This function is identical to NRZ-I.

Bi-Phase-L (Manchester)

In: Bit stream

Out: Bit stream



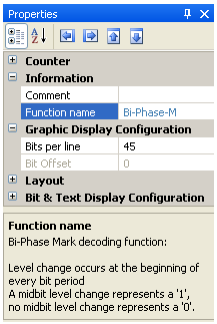
Function:

Analyzes the bit changes of the bit stream. A change from '1' to '0' represents a '1' and a change from '0' to '1' represents a '0'. The bits are analyzed in pairs, i.e. the number of output bits is half the number of input bits.

Bi-Phase-M

In: Bit stream

Out: Bit stream



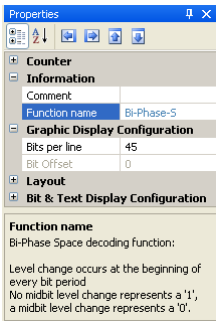
Function:

In Bi-Phase-M encoding, a logical '1' is represented by a pair of bits of opposite values ('10' or '01'). A logical '0' is represented by a pair of bits of the same values ('00' or '11'). The decoding procedure halves the number of output bits.

Bi-Phase-S

In: Bit stream

Out: Bit stream



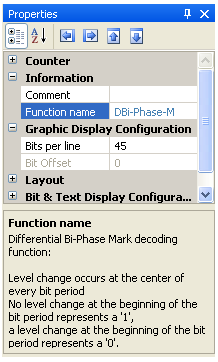
Function:

In Bi-Phase-S encoding, a logical '0' is represented by a pair of two bits of opposite values ('10' or '01'). A logical '1' is represented by a pair of bits of the same value ('00' or '11'). The decoding procedure halves the number of output bits.

DBi-Phase-M

In: Bit stream

Out: Bit stream



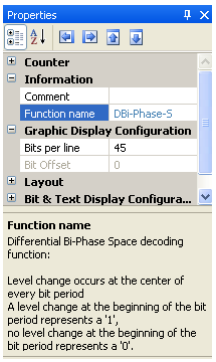
Function:

Two bits form a bit period. A bit change at the beginning of a bit period represents a '0', while no bit change at the beginning of a bit period represents a '1'.

DBi-Phase-S

In: Bit stream

Out: Bit stream



Function:

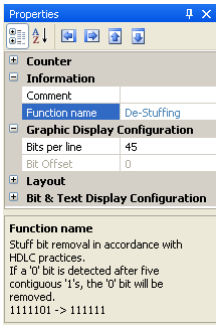
Two bits form a bit period. A bit change at the beginning of a bit period represents a '1', while no bit change at the beginning of a bit period represents a '0'.

Bit Manipulation

De-Stuffing (HDLC)

In: Bit stream

Out: Bit stream



Function:

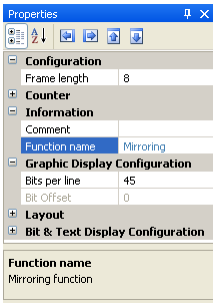
Removes stuff bits inserted in the input bit stream. If a zero bit is detected after five contiguous Ones, the Zero bit will be removed.

Example: "1111101" is changed to " '111111"

Mirroring

In: Bit stream

Out: Bit stream



Function:

The mirroring function modifies the incoming bit stream frame by frame. The **Frame length** is user defined. The function changes the bit order within each frame.

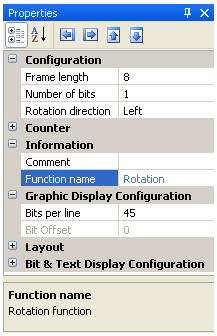
Example with a frame size of 5 bits:

"10111 01101" is changed to "11101 10110"

Rotation

In: Bit stream

Out: Bit stream



Function:

The rotation function modifies the incoming bit stream frame by frame. The **Frame length** is user defined, as well as the **Rotation direction** and the **Number of bits** to be rotated.

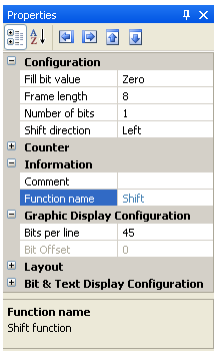
Example with a frame size of 5 bits, left rotation direction and one rotation step:

"10111 01101" is changed to "01111 11010"

Shift

In: Bit stream

Out: Bit stream



Function:

The shift function modifies the incoming bit stream frame by frame. The **Frame length** is user defined, as well as the **Shift direction**, the **Number of bits** to be shifted and the **Fill bit value**.

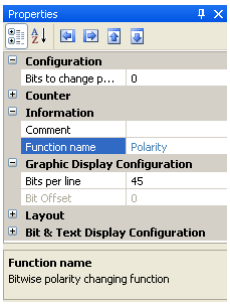
Example with a frame size of 5 bits, left shift direction, two bits shift and a fill value of '1':

"10111 01101" is changed to "11111 10111"

Polarity

In: Bit stream

Out: Bit stream



Function:

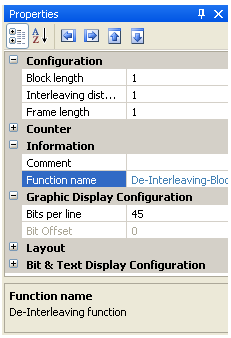
Inverts the bit stream according to the **Bits to change polarity** pattern. At positions marked with a One, the bit is inverted, at positions with a Zero, the bit remains unchanged.

Example:	#1	#2	#3
Bits to change polarity	111000111	1	0
Input	111111111000000000	111111111000000000	111111111000000000
Output	000111000111000111	000000000111111111	111111111000000000

De-Interleave Bit Block

In: Bit stream

Out: Bit stream



Function:

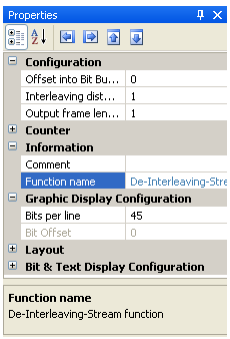
Changes the bit order according to the settings of **Block length**, **Frame length** and **Interleaving distance**. The easiest way to understand the de-interleaving function is a closer look at the examples below (imagine that the bit stream is written horizontally into the buffer and read out vertically):

Example:	#1	#2
Distance	3	2
Block length	12	16
Frame length	1	2
Input	000111000111	0011001111100010
Matrix	000 111 000 111	00 11 00 11 11 10 00 10
Output	010101010101	0000110011111010

De-Interleaving Stream

In: Bit stream

Out: Bit stream



Function:

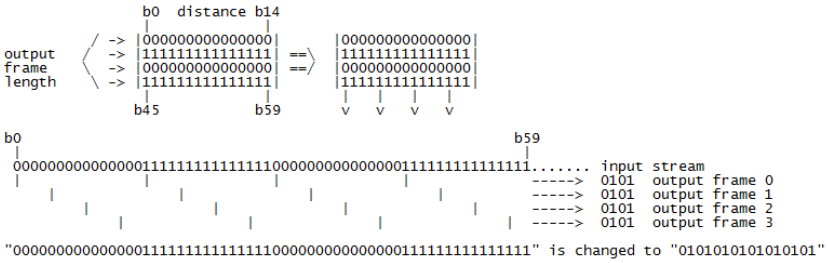
Change the bit order according to the settings of **Offset into Bit Buffer**, **Output frame length** and **Interleaving distance**. The **Offset into Bit Buffer** tells the function where to start the de-interleaving function. Is **Offset into Bit Buffer** for example set to 3, then the first 3 bits will not be used for calculation of the output data. According to the **Output frame length** setting, the output data will be less than the input data. The easiest way to understand the de-interleaving stream function is a closer look to an example below (imagine that the bit stream is written horizontally into the buffer and read out vertically):

Example 1:

Offset into Bit Buffer = 0

Interleaving distance = 15

Output frame length = 4

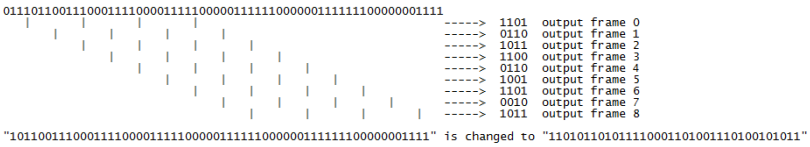


Example 2:

Offset into Bit Buffer = 3

Interleaving distance = 8

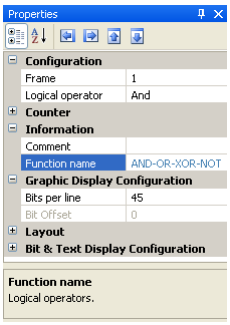
Output frame length = 4



AND / OR / XOR / NOT

In: Bit stream

Out: Bit stream



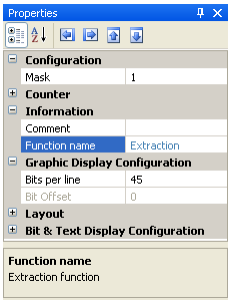
Function:

The output bit values depend on the selected logical operation (**Logical operator**) peformed on the input bits. The first operand is the input bit stream, while the second operand (**Frame**) is constant and can be '0' or '1'.

Extraction

In: Bit stream

Out: Bit stream



Function:

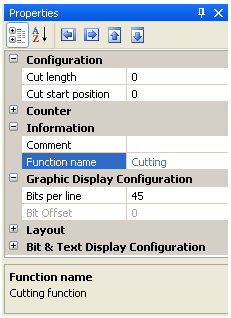
Extracts bits from the incoming bit stream using the user defined **Mask**. Only positions marked with a '1' are extracted. The output bit stream is calculated frame by frame.

Example: "111110" with mask "110" changes to "1111"

Cutting

In: Bit stream

Out: Bit stream



Function:

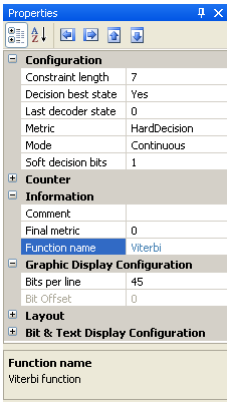
Cuts **Cut length** bits, beginning at **Cut start position**. Note that counting starts at zero, i.e. the first element in the bit stream is number 0.

Decoding/Equalizer

Viterbi-Decoding

In: Bit stream

Out: Bit stream



Function:

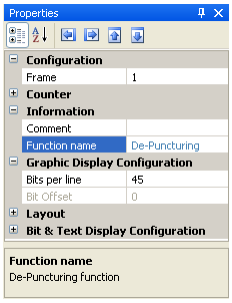
Decodes the incoming bit stream using the Viterbi algorithm - a maximum-likelihood decoding procedure for convolutional codes.

Parameter	
Constraint length	Equals n+1, where n is the length of the constraint register in the encoder
Decision best state	Use best state or not
Last decoder state	Initial state of the decoder
Metric	Select hard or soft decision
Mode	Select whether the input data should be treated as a continuous stream or a stream of bursts.
Soft decision bits	If soft decision is used, enter the number of soft decision bits

De-Puncturing

In: Bit stream

Out: Bit stream



Function:

This function adds probability bits and de-puncturing bits to the input bit stream.

Parameter	Value
Frame	According to this bit pattern, bits are inserted. The length of the bit pattern corresponds to the frame length. At positions marked with a '0', a bit is inserted. Additionally, for every input bit, a probability bit is added. For received bits (marked with a '1' in the frame pattern), a '1' probability bit is added - for inserted bits, a '0' probability bit is added (equals a probability of 0.5)

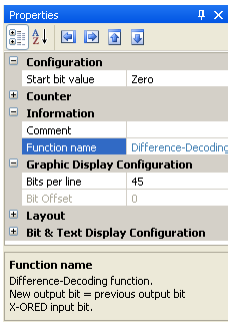
Example with a frame pattern of "110":

The frame pattern "110" means that after two input bits, a de-puncturing bit must be inserted so "1111" becomes "111100111100".

Difference-Decoding

In: Bit stream

Out: Bit stream



Function:

This function performs difference decoding, which is a logical XOR operation on the previous output bit and the current input bit. The **Start bit value** is user defined.

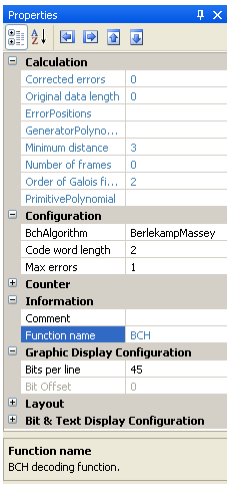
Example with start bit value of 'One':

“01101110” is changed to “10110100”

BCH-Decoding

In: Bit stream

Out: Bit stream



Function:

Decodes BCH encoded bit streams.

Parameter	Value
Code word length	Length of code word including redundancy bits.
Max errors	Error correction capability

Example with code length 15 and error correction capability 3:

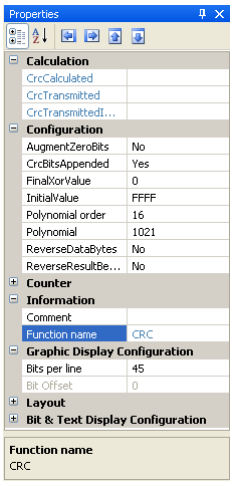
“011001010000111” is changed to “00111”

CRC & Polynomial

CRC (1..32)

In: Bit stream

Out: Bit stream



Function:

Calculate the cyclic redundancy checksum (CRC) value of the input bit stream according to the settings described below.

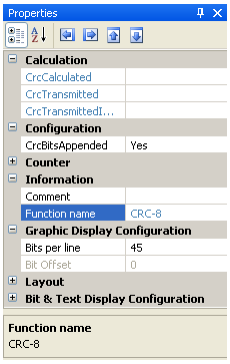
Parameter	Value
AugmentZeroBits	Calculate CRC with or without augmented (added) zero bits
CrcBitsAppended	Are CRC bits appended to the bit stream (**Yes** or **No**)
FinalXorValue	Final XOR value in hex
InitialValue	Initial value in hex
Order	Polynomial order (1..32)
Polynomial	Polynomial in hex
ReverseDataBytes	Reflect data byte before processing (**Yes** or **No**)
ReverseResultBeforeXor	Reflect final result before XOR (**Yes** or **No**)

The calculated CRC value is displayed in the 'CrcCalculated' field. If 'CrcBitsAppended' is set to 'Yes', then the 'CrcTransmitted' field contains the transmitted CRC value in hex form. If there are no appended CRC bits, then the 'CrcTransmitted' field has no meaning. The transmitted CRC value is displayed in inversed form too ('CrcTransmittedInverse').

CRC-8

In: Bit stream

Out: Bit stream



Function:

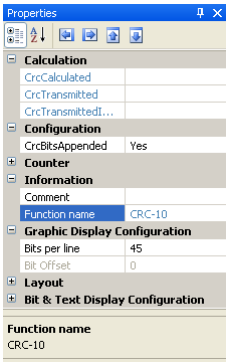
Calculates the standard CRC-8 values of the incoming bit stream.

The calculated CRC value is displayed in the **CrcCalculated** field. If **CrcBitsAppended** is set to **Yes**, then the **CrcTransmitted** field contains the transmitted CRC value in hex form. If there are no appended CRC bits, then the **CrcTransmitted** field has no meaning. The transmitted CRC value is alsodisplayed in inverted form (**CrcTransmittedInverse**).

CRC-10

In: Bit stream

Out: Bit stream



Function:

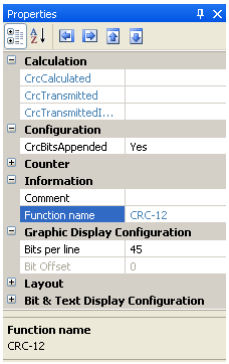
Calculates the standard CRC-10 values of the incoming bit stream.

The calculated CRC value is displayed in the **CrcCalculated** field. If **CrcBitsAppended** is set to **Yes**, then the **CrcTransmitted** field contains the transmitted CRC value in hex form. If there are no appended CRC bits, then the **CrcTransmitted** field has no meaning. The transmitted CRC value is alsodisplayed in inverted form (**CrcTransmittedInverse**).

CRC-12

In: Bit stream

Out: Bit stream



Function:

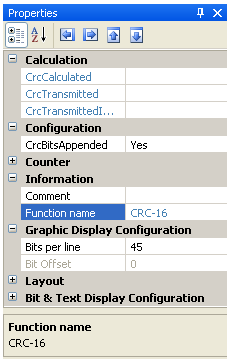
Calculates the standard CRC-12 values of the incoming bit stream.

The calculated CRC value is displayed in the **CrcCalculated** field. If **CrcBitsAppended** is set to **Yes**, then the **CrcTransmitted** field contains the transmitted CRC value in hex form. If there are no appended CRC bits, then the **CrcTransmitted** field has no meaning. The transmitted CRC value is alsodisplayed in inverted form (**CrcTransmittedInverse**).

CRC-16

In: Bit stream

Out: Bit stream



Function:

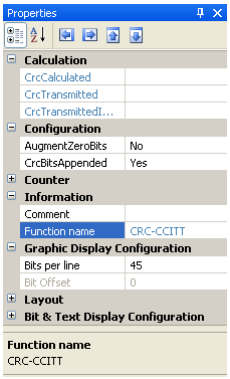
Calculates the standard CRC-16 values of the incoming bit stream.

The calculated CRC value is displayed in the **CrcCalculated** field. If **CrcBitsAppended** is set to **Yes**, then the **CrcTransmitted** field contains the transmitted CRC value in hex form. If there are no appended CRC bits, then the **CrcTransmitted** field has no meaning. The transmitted CRC value is alsodisplayed in inverted form (**CrcTransmittedInverse**).

CRC-CCITT

In: Bit stream

Out: Bit stream



Function:

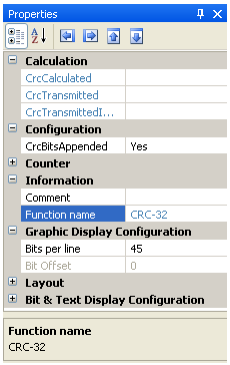
Calculate the standard CRC-CCITT values of the incoming bit stream.

The calculated CRC value is displayed in the **CrcCalculated** field. If **CrcBitsAppended** is set to **Yes**, then the **CrcTransmitted** field contains the transmitted CRC value in hex form. If there are no appended CRC bits, then the **CrcTransmitted** field has no meaning. The transmitted CRC value is alsodisplayed in inverted form (**CrcTransmittedInverse**).

CRC-32

In: Bit stream

Out: Bit stream



Function:

Calculates the standard CRC-32 values of the incoming bit stream.

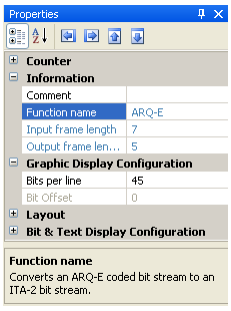
The calculated CRC value is displayed in the **CrcCalculated** field. If **CrcBitsAppended** is set to **Yes**, then the **CrcTransmitted** field contains the transmitted CRC value in hex form. If there are no appended CRC bits, then the **CrcTransmitted** field has no meaning. The transmitted CRC value is alsodisplayed in inverted form (**CrcTransmittedInverse**).

Channel Decoding (Protocol)

ARQ-E

In: Bit stream

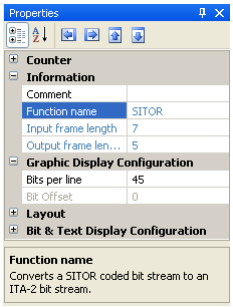
Out: Bit stream



Function:
Converts an ARQ-E coded bit stream into an ITA-2 bit stream.

SITOR

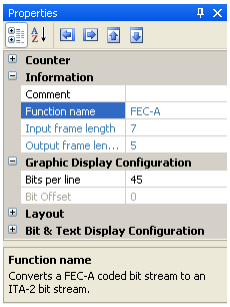
In: Bit stream
Out: Bbit stream



Function:
Converts an SITOR coded bit stream into an ITA-2 bit stream.

FEC-A

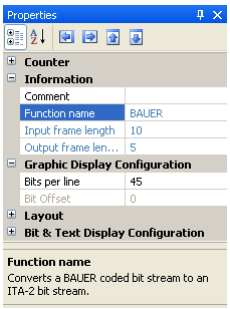
In: Bit stream
Out: Bit stream



Function:
Converts an FEC-A coded bit stream into an ITA-2 bit stream.

BAUER

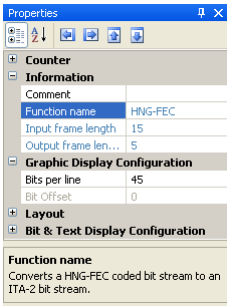
In: Bit stream
Out: Bit stream



Function:
Converts a BAUER coded bit stream into an ITA-2 bit stream.

HNG-FEC

In: Bit stream
Out: Bit stream



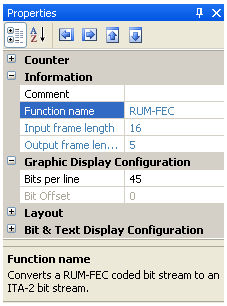
Function:

Converts an HNG-FEC coded bit stream into an ITA-2 bit stream.

RUM-FEC

In: Bit stream

Out: Bit stream



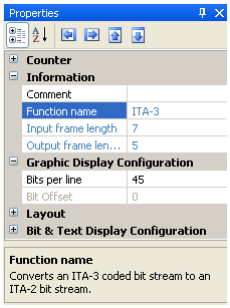
Function:

Converts an RUM-FEC coded bit stream into an ITA-2 bit stream.

ITA-3 (M.342)

In: Bit stream

Out: Bit stream



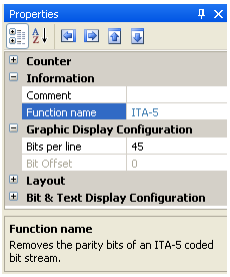
Function:

Converts an ITA-3 coded bit stream into an ITA-2 bit stream.

ITA-5

In: Bit stream

Out: Bit stream



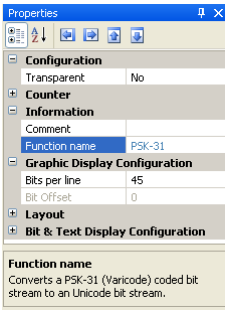
Function:

Removes the parity bits of an ITA-5 coded bit stream.

PSK-31

In: Bit stream

Out: Bit stream



Function:

Converts a PSK-31 (Varicode) code into an ITA-2 bit stream.

Source Decoding (Alphabet)

Latin

In: Bit stream

Out: Character stream

Function:

Converts a bit stream to Unicode text (Latin). If **Transparent** display mode is set to **No,** then special characters like carriage return or line feed are treated as control characters and applied to the output window. If **Transparent** display mode is set to **Yes**, then the special characters are displayed using a corresponding, descriptive symbol.

The source code format can be set to ITA-2 or ITA-1.

Third-shift Greek

In: Bit stream

Out: Character stream

Function:

Converts a bit stream to Unicode text (Third-shift Greek). If **Transparent** display mode is set to **No**, then special characters like carriage return or line feed are treated as control characters and applied to the output window. If **Transparent** display mode is set to **Yes**, then the special characters are displayed using a corresponding, descriptive symbol.

Cyrillic

In: Bit stream

Out: Character stream

Function:

Converts a bit stream to Unicode text (Cyrillic). If **Transparent** display mode is set to **No**, then special characters like carriage return or line feed are treated as control characters and applied to the output window. If **Transparent** display mode is set to **Yes**, then the special characters are displayed using a corresponding, descriptive symbol.

Tass-Cyrillic

In: Bit stream

Out: Character stream

Function:

Converts a bit stream to Unicode text (TASS Cyrillic). If **Transparent** display mode is set to **No**, then special characters like carriage return or line feed are treated as control characters and applied to the output window. If **Transparent** display mode is set to **Yes**, then the special characters are displayed using a corresponding, descriptive symbol.

Third-shift Cyrillic

In: Bit stream

Out: Character stream

Function:

Converts a bit stream to Unicode text (Third-shift Cyrillic). If **Transparent** display mode is set to **No**, then special characters like carriage return or line feed are treated as control characters and applied to the output window. If **Transparent** display mode is set to **Yes**, then the special characters are displayed using a corresponding, descriptive symbol.

Hebrew

In: Bit stream

Out: Character stream

Function:

Converts a bit stream to Unicode text (Hebrew). If **Transparent** display mode is set to **No**, then special characters like carriage return or line feed are treated as control characters and applied to the output window. If **Transparent** display mode is set to **Yes**, then the special characters are displayed using a corresponding, descriptive symbol.

Arabic Baghdad-70

In: Bit stream

Out: Character stream

Function:

Converts a bit stream to Unicode text (Arabic Baghdad-70). If **Transparent** display mode is set to **No**, then special characters like carriage return or line feed are treated as control characters and applied to the output window. If **Transparent** display mode is set to **Yes**, then the special characters are displayed using a corresponding, descriptive symbol.

Arabic Baghdad-80 (ATU-80)

In: Bit stream

Out: Character stream

Function:

Converts a bit stream to Unicode text (Arabic Baghdad-80). If **Transparent** display mode is set to **No**, then special characters like carriage return or line feed are treated as control characters and applied to the output window. If **Transparent** display mode is set to **Yes**, then the special characters are displayed using a corresponding, descriptive symbol.

Bulgarian

In: Bit stream

Out: Character stream

Function:

Converts a bit stream to Unicode text (Bulgarian). If **Transparent** display mode is set to **No**, then special characters like carriage return or line feed are treated as control characters and applied to the output window. If **Transparent** display mode is set to **Yes**, then the special characters are displayed using a corresponding, descriptive symbol.

Swedish

In: Bit stream

Out: Character stream

Function:

Converts a bit stream to Unicode text (Swedish). If **Transparent** display mode is set to **No**, then special characters like carriage return or line feed are treated as control characters and applied to the output window. If **Transparent** display mode is set to **Yes**, then the special characters are displayed using a corresponding, descriptive Symbol.

Danish-Norwegian

In: Bit stream

Out: Character stream

Function:

Converts a bit stream to Unicode text (Danish/Norwegian). If **Transparent** display mode is set to **No**, then special characters like carriage return or line feed are treated as control characters and applied to the output window. If **Transparent** display mode is set to **Yes**, then the special characters are displayed using a corresponding, descriptive symbol.

German

In: Bit stream

Out: Character stream

Function:

Converts a bit stream to Unicode text (German). If **Transparent** display mode is set to **No**, then special characters like carriage return or line feed are treated as control characters and applied to the output window. If **Transparent** display mode is set to **Yes**, then the special characters are displayed using a corresponding, descriptive symbol.

French

In: Bit stream

Out: Character stream

Function:

Converts a bit stream to Unicode text (French). If **Transparent** display mode is set to **No**, then special characters like carriage return or line feed are treated as control characters and applied to the output window. If **Transparent** display mode is set to **Yes**, then the special characters are displayed using a corresponding, descriptive symbol.

US

In: Bit stream

Out: Character stream

Function:

Converts a bit stream to Unicode text (US). If **Transparent** display mode is set to **No**, then special characters like carriage return or line feed are treated as control characters and applied to the output window.

If **Transparent** display mode is set to **Yes**, then the special characters are displayed using a corresponding, descriptive symbol.

ASCII

In: Bit stream

Out: Character stream

Function:

Converts a bit stream to ASCII text. The user can select between 7 bit ASCII and 8 bit ASCII.

UNICODE

In: Bit stream

Out: Character stream

Function:

Converts a bit stream to Unicode text. The user can select between Little- and Big-Endian.

UTF-7

In: Bit stream

Out: Character stream

Function:

Converts a bit stream to UTF-7 text.

UTF-8

In: Bit stream

Out: Character stream

Function:

Converts a bit stream to UTF-8 text.

Analysis Tools

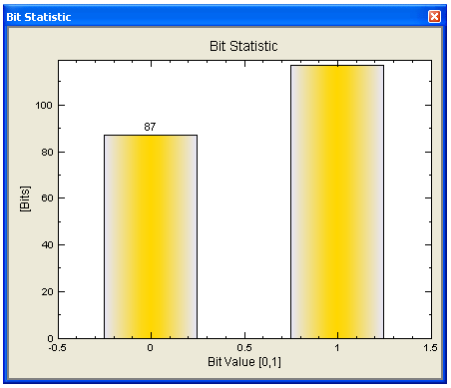
Bit Statistic

In: Bit stream

Out: Chart

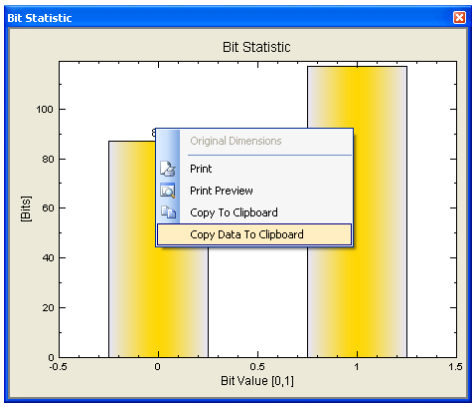
Function:

Generates a bit histogram containing the statistical distribution of logical zeros and ones.



This chart is only calculated once, i.e. the content does not change, even if the analysis set is recalculated. To update histogram values, close the window and then reopen it.

A right click on the histogram makes additional functionality available.



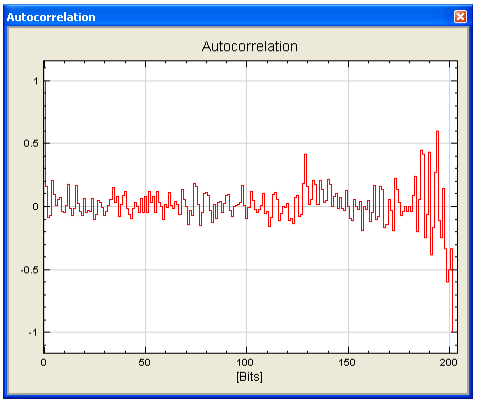
Autocorrelation

In: Bit stream

Out: Chart

Function:

Generates a graphical display of an autocorrelation operation on the input bit stream.

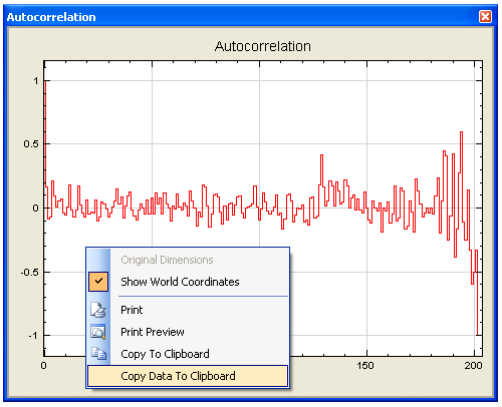


This chart is only calculated once, i.e. the content does not change, even if the analysis set is recalculated. To update graph values, close the window and then reopen it.

Built-in zooming functions that are available using mouse clicks.

A drag-and-drop operation will select an area for zooming.

A right click on the display makes additional functionality available.



If the view has been changed by drag and drop, the original dimensions can be restored by clicking **Original Dimensions**.

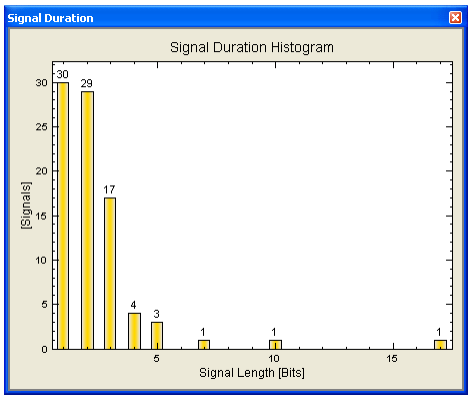
Signal Duration

In: Bit stream

Out: Histogram

Function:

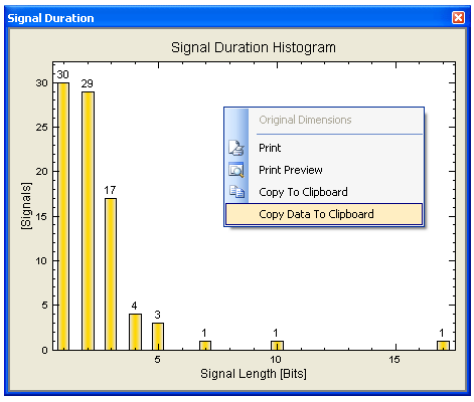
Generates a statistical histogram of signal duration.



This chart is only calculated once, i.e. the content does not change, even if the analysis set is recalculated. To update histogram values, close the window and then reopen it.

A drag-and-drop operation will select an area for zooming.

A right click on the display makes additional functionality available.



If the view has been changed by drag and drop, the original dimensions can be restored by clicking **Original Dimensions**.

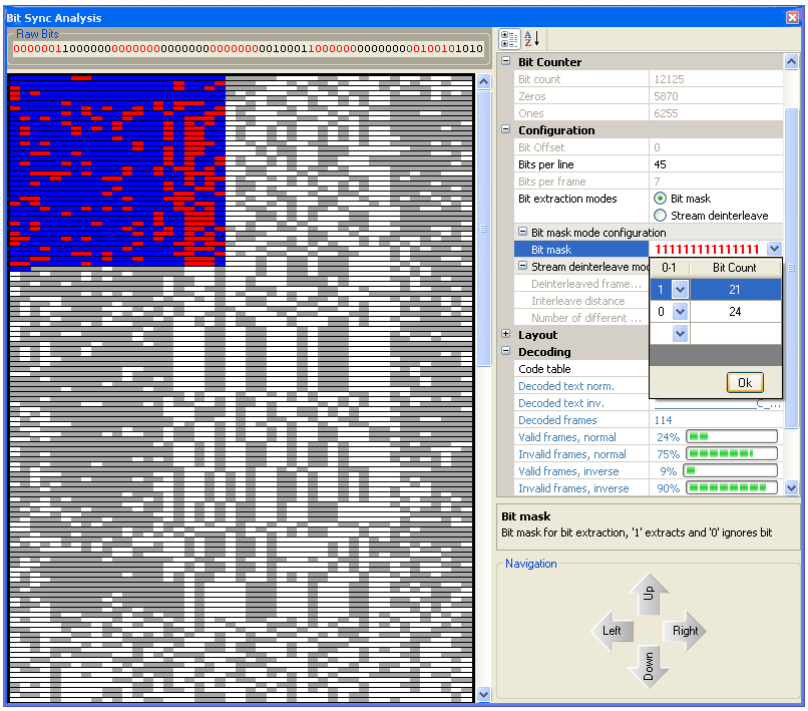
Bit Sync Analysis

In: Bit stream

Out: Analysis window

Function:

Opens a bit synchronization analysis window.



Bit Sync Analysis is designed to find the starting position of a frame. For this reason, all bits are displayed in a graphical view with an adjustable number of bits per line; this makes it easier to find periodic sequences.

The settings for **Bit Sync Analysis** are defined in the **Properties** window:

Bits per line defines the number of bits per line.

Bit extraction mode offers a choice between **Bit mask** and **Stream deinterleave** modes. Depending on the mode selected, either the bit mask or the de-interleaving can be configured. For details of these functions, please refer to the descriptions of the **Extraction** and **De-Interleaving Bit Stream** functions in the **Bit Manipulation** section of this manual.

The **Layout** category controls the appearance of the graphical bit display.

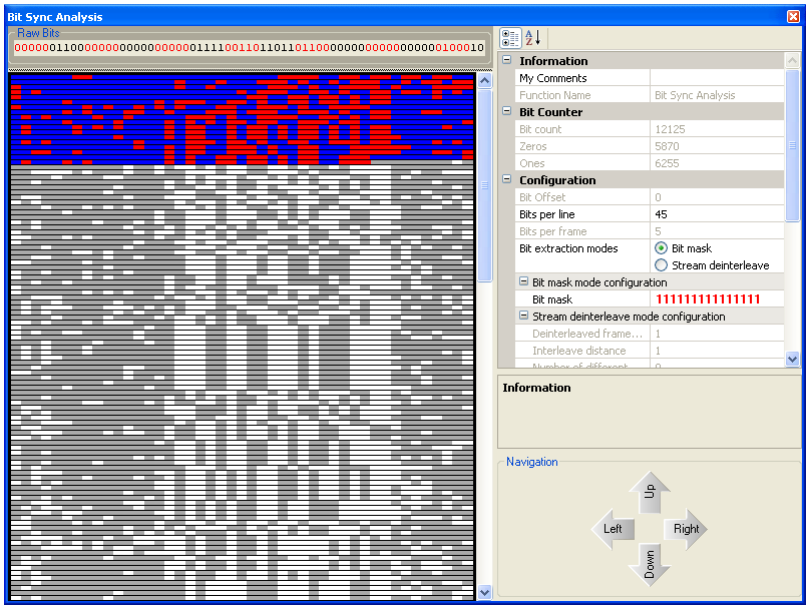
In the **Decoding** category, the decoding alphabet is selected. Supported alphabets are ITA2, ITA3, CCIR476_5, ASCII (7 Bit) and ASCII (8 Bit). Decoding is only possible if **Bit mask** is activated.

The CCIR476_5 and ITA3 alphabets are redundant alphabets allowing a calculation of the number of valid frames, which is displayed in the decoding category. This is an additional help to find the start of a frame inside a bit stream. For the other alphabets, a validation is not possible.

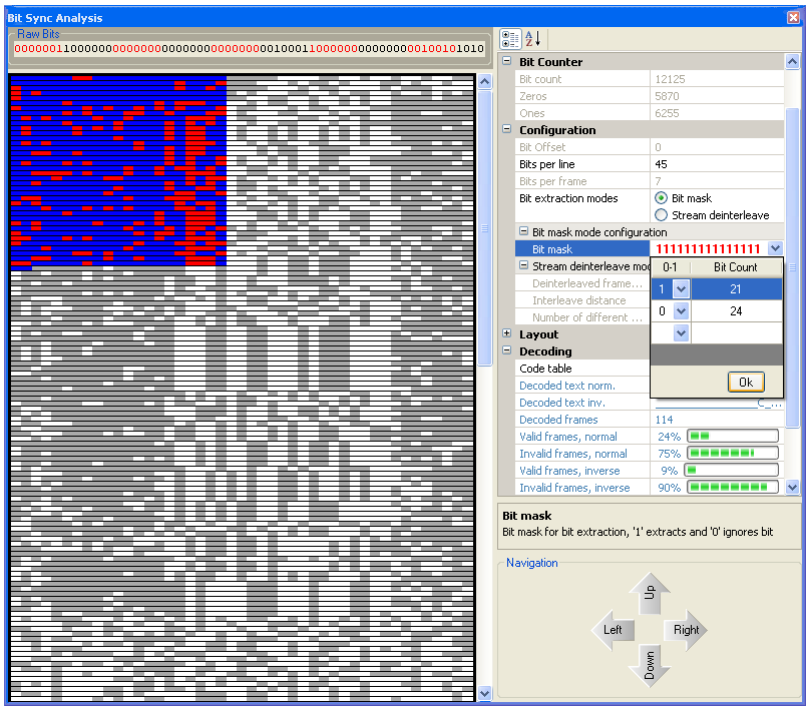
The **Navigation** category defines the behavior of the four arrow buttons in the bottom area of the dialog window.

Example:

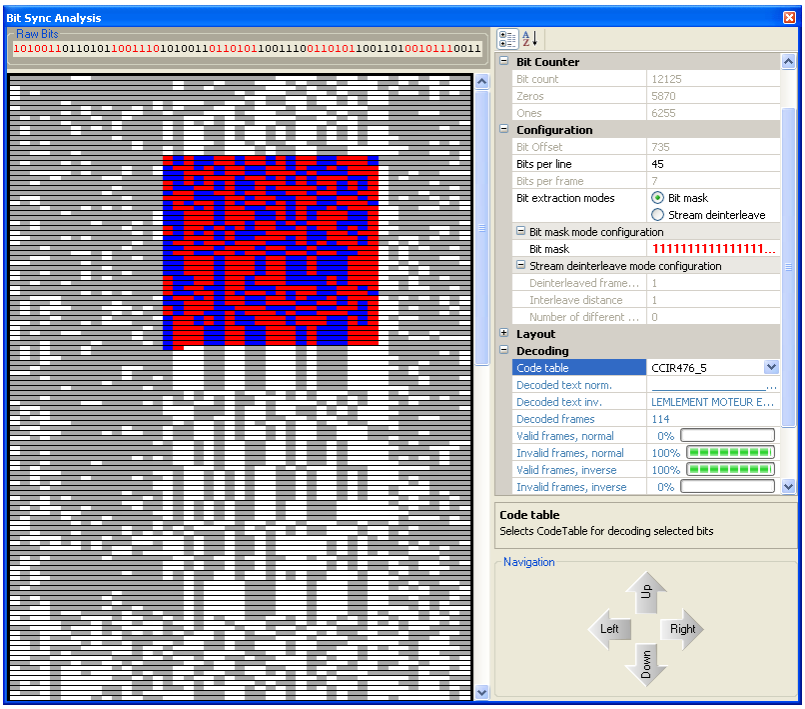
Open the SITOR example analysis, located in the **BitViewTool\Examples** folder and select the **Import IAS Bit stream** function in the **History Explorer**. Open the **Analysis Tools** in the **Toolbox** and select **Bit Sync Analysis**. The dialog below will appear:



SITOR-A has a block length of 45, so adjust **Bits per line** to 45 bits. As the alphabet is known, select **CCIR476_5**. The next step is to configure the **Bit mask** with 21 ones and 24 zeros. After configuration of the bit mask, the dialog looks like this:



As the SITOR alphabet allows validation, the remaining task is to move the selection across the window with the navigation arrow buttons and check the percentage of valid frames. The figure below shows a valid bit block:



The decoded text is displayed in the **Decoding** category.

Custom Library

Want to Roll Your Own functions?

The ability to expand BitViewTool with custom developed functions is one of the most powerful features of this application. However, before you start rolling your own functions you must have a good grasp of C#, object oriented programming and the use of Microsoft Visual Studio and .NET. In addition a solid knowledge of Matlab and the mathematical aspects of communication systems is a prerequisite to benefit from these powerful tools. The books listed below may help the programmer to get acquainted with the development tools,

- Bernhard Sklar, **"Digital Communications: Fundamentals and Applications"**, 2nd Ed., Prentice-Hall, 2001
- John Sharp, **"Microsoft Visual C# 2005 – Step by Step"**, Microsoft Press, 2006
- Ruda Pratap, **"Getting Started with MATLAB 7"**, Oxford

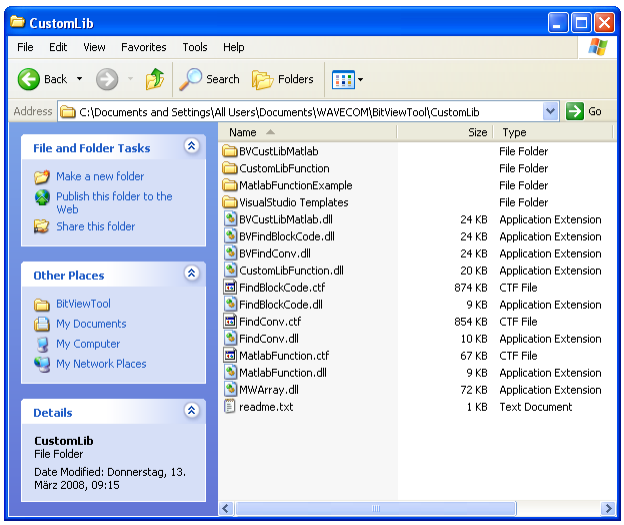
The Custom Library Interface supports the integration of third party functions into BitViewTool. Functions may be implemented using any .NET language, i.e. C#.NET, VB.NET, J#.NET and C++.NET, using the .NET Framework 2.0. Custom functions are compiled into individual 32-bit **.NET-DLLs** and executed on operating systems supported by BitViewTool, including Windows 2000, XP, 2003 Server and Vista.

As from release 2.1 it is also possible to write mathematical functions in Matlab and integrate them with BitViewTool.

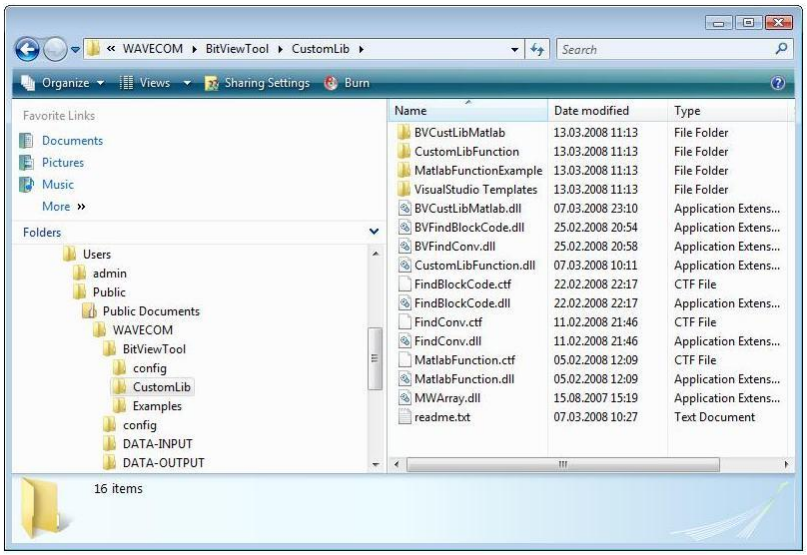
Examples using custom functions are distributed with BitViewTool. The custom library DLL as well as the source code can be found in the *CustomLib* folder of the BitViewTool. The projects (solutions) provided were created using VS.NET 2005 with .NET Framework 2.0.

The *CustomLib* folder is created during installation.

On Windows XP and older:



On Windows Vista:



Four subfolders are also created:

CustomLibFunction

A .NET custom library example. The output of this project is *CustomLibFunction.dll*, which can be used with BitViewTool.

BVCustLibMatlab

Example project for a wrapper for a Matlab custom function. The output of this project is *BVCustLibMatlab.dll.*,

MatlabFunctionExample

Example project for Matlab. This is an example that shows how to compile Matlab code into .NET code. The output of this project is *MatlabFunction.dll* and *MatlabFunction.ctf*. Together with *BVCustLibMatlab.dll*, these files build a Matlab custom function that can be used with BitViewTool.

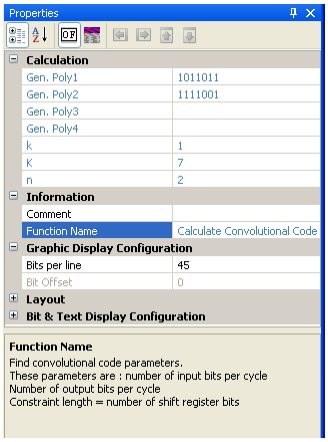
VisualStudio Templates

Two templates for Microsoft Visual Studio are provided to create new custom function projects. Please, read the *Readme.txt* file that is included in this subfolder for more information.

As examples and to demonstrate the Matlab custom functions, two additional functions (without project and source files) are included:

Calculate Convolutional Code (**FindConv**) is used to find the parameters of convolutional codes. The function returns constraint length (K), number of input bits per shift cycle (k), number of output bits per shift cycle (n) and generator polynomials.

The function will search for K = 2..14, and n = 2..4. The number of returned generator polynomials depend on the number of output bits per shift cycle (n).

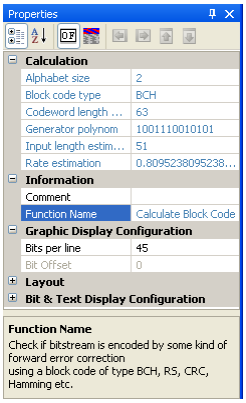


Calculate Block Code (**FindBlockCode**) is used to identify forward error correction codes like BCH, RS, CRC or Hamming. The function returns alphabet size (normally equal to 2 because input is a binary bit stream), block code type (BCH, RS, etc.), codeword length estimation (n), generator polynomial, input length estimation (k) and code rate estimation (R = k/n).

The function will return the following **Block Code Types,**

Unidentified code
BCH
CRC or perfect cyclic code
Binary repetition (reversals)
Binary Golay
Binary Hamming
CRC block code
Non-cyclic block code

The generator polynomial is returned as a string of ones and zeros, starting at the lowest order of 2^x, e.g 1001110010101 means 1 + x^3 + x^4 + x^5 + x^8 + x^10 + x^12.



Adding a Custom Function

Adding a new custom function is simply done by adding its DLL to the **CustomLib** folder.

Windows XP and older:

- Documents and Settings\All Users\Documents\WAVECOM\BitViewTool\CustomLib

or

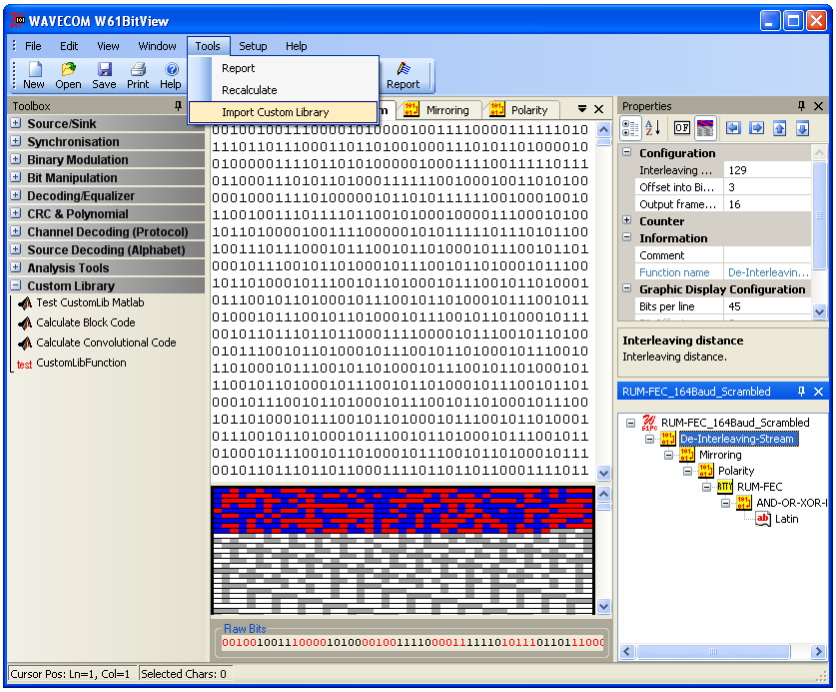
- Documents and Settings\All Users\Shared Documents\WAVECOM\BitViewTool\CustomLib

Windows Vista:

- Users\Public\Public Documents\WAVECOM\BitViewTool\ CustomLib

BitViewTool automatically adds the function to the Custom Library list in the Toolbox after launch. Alternatively, if BitViewTool is already running, the custom function may be added to the list using the **Import Custom Library** item in the **Tools** menu.

Please, be aware that the Custom Library button will not appear in the Toolbox if a custom function is not provided.



The icon file for the custom function is named *CustomLibFunction.bmp* and can be customized using the built-in editor in Visual Studio. By compiling the custom function, the icon is integrated into the custom lib DLL.

Constraints

All BitViewTool functions belong to one of three groups:

- Source functions, which have no input.
- Sink functions, which have no output.
- General functions, which have both input and output, and where the input type is *BitArray* and the output type may be *BitArray* or *string*.

Presently custom functions are limited to be general functions with the input and output type is *BitArray*.

Important Notes

When creating custom functions, certain rules must be observed. Please, refer to the source code templates and examples for more details.

The following source code elements (names) are mandatory and may not be modified:

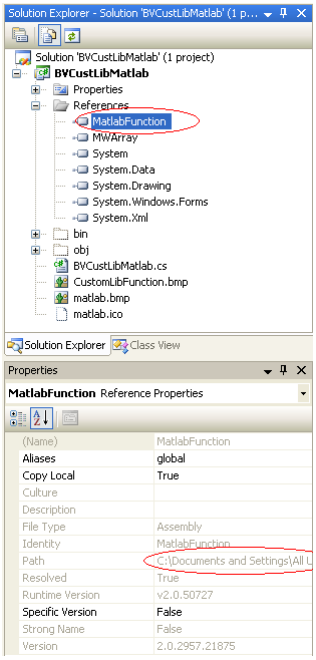
- namespace *CustomLib*
- public string *FunctionName*
- public string *Comment*
- public BitArray *Calculate(BitArray buf)*

Property categories and additional properties may be defined without limitations.

Steps to Write a Custom Function in C# .NET

- Open the *CustomLibFunction* example or create a new project based on the *BVCustLibFunc* template.
- Change assembly and class name according to your needs.
- Modify the properties and expand the *calculate* function with your code.
- Compile the project and copy the output DLL to the *CustomLib* folder (see above)

Steps to Write a Custom Function with Matlab



- Open the *MatlabFunctionExample* project (*MatlabFunction.prj*) in Matlab.
- Change the .M-File according to your needs.
- Compile the Matlab code. If the compilation process was successful, there will be two new files in the *.\MatlabFunction\distrib* subfolder, *MatlabFuntion.dll* and *MatlabFunction.ctf*.
- Open the *BVCustLibMatlab* example or create a new project based on the *BVCustLibMatlab* template.
- Change assembly and class name according to your needs.
- Set a reference to the *MatlabFunction.dll* in the *\distrib* subfolder. This way, the *BVCustLibMatlab* class will know where to find the Matlab DLL.
- Compile the *BVCustLibMatlab* project and copy all the files in the output directory to the *CustomLib* folder.

Source Code Template / Example (C# .NET)

CustomLibFunction.cs

```
//-------------------------------------------------------------------------------
// File        : CustomLibFunction.cs
// Author      : Wavecom Elektronik AG
// Date        : February 2008
// Description : Template/example for a custom defined library function.
//
//               Important note:
//               In the project settings the "Assembly Name" must be
//               equal to the "Default Namespace".
//
//-------------------------------------------------------------------------------

#region using references

using System;
using System.Collections;
using System.Text;
using System.ComponentModel;
using CustomLibFunction.Properties;

#endregion

namespace CustomLib  // Mandatory! Do not change this name.
{
    /// <summary>
    /// </summary>
    [Serializable]
    [DefaultProperty("FunctionName")] // adjust string if necessary, it points to a property
below
    public class CustomLibFunction
    {
        #region Constant fields

        private class PropertyCategory
        {
            // these are the categories in the BitViewTool property grid (Parameter Window)
            public const string Information = "Information";
            public const string Counter = "Counter";
            public const string Configuration = "Configuration";
            public const string Calculation = "Calculation";

            // do not change these names, they are used to distinguish the different catego-
ries
            // when any change in the parameters happens
        }

        private const int MAX_SIZE = 500000; // do not modify this value

        #endregion

        #region Fields

        // generate a new BitArray for calculation, it will be returned to the calling func-
tion
        private BitArray outbox = new BitArray(MAX_SIZE);

        #endregion

        #region Mandatory Properties

        private string functionName = "CustomLibFunction";
        /// <summary>
        /// Mandatory! Do not delete this property! The string "CustomLibFunction" may be
modified.
        /// This will be the name of the Function in the BitViewTool History explorer and
ToolBox
        /// </summary>
        [Category(PropertyCategory.Information)]
        [Description("Description of CustomLibFunction goes here.\n" +
```

```
                    "Here: example of a custom defined library function. Inverts all the
input bits.\n")]
        [DisplayName("Function Name")]
        public string FunctionName
        {
            get
            {
                return (functionName);
            }
        }


        private string comment = string.Empty;
        /// <summary>
        /// Mandatory! Do not modify this property!
        /// </summary>
        [Category(PropertyCategory.Information)]
        [Description("My comments.")]
        [DefaultValue("")]
        public string Comment
        {
            get
            {
                return comment;
            }
            set
            {
                comment = value;
            }
        }

        private static System.Drawing.Image iconImage = Resources.CustomLibFunction;
        /// <summary>
        /// If available, you can specify a custom image displayed on the function button
        /// in the BitView toolbox. The image must be 16x16 pixel. The bitmap file must be
        /// imported into the CustomLibFunction resources, so that it can be refered to it as
shown above!
        /// Currently the image is the .bmp file 'CustomFunction.bmp'. It is part of this so-
lution, it can be
        /// seen in the Solution Explorer Window and it can be modified.
        /// If there is no image available, remove all private and public fields, i.e. iconI-
mage and IconImage,
        /// iconTransparentColor and IconTransparentColor
        /// below. A default icon image is then added by the BitViewTool main application.
        /// </summary>
        [Category(PropertyCategory.Information)]
        [Description("Icon bitmap for Toolbox Button and History Explorer")]
        [Browsable(false)]
        public static System.Drawing.Image IconImage
        {
            get
            {
                return iconImage;
            }
        }

        private     static     System.Drawing.Color     iconTransparentColor     =     Sys-
tem.Drawing.Color.White;
        /// <summary>
        /// If available, you can specify a custom image display on the function button
        /// in the toolbox. The image must be 16x16 pixel, the transparent color must
        /// be specified here
        /// </summary>
        [Category(PropertyCategory.Information)]
        [Description("Icon bitmap transparent color")]
        [Browsable(false)]
        public static System.Drawing.Color IconTransparentColor
        {
            get
            {
                return iconTransparentColor;
            }
        }

        #endregion

        #region Optional Properties
```

```
        private string dummyCalculation = string.Empty;
        /// <summary>
        /// Optional result form this function! DummyCalculation. Put all results into this
category
        /// </summary>
        [Category(PropertyCategory.Calculation)]
        [Description("Description of DummyCalculation goes here. Here: number of output
bits.")]
        [DisplayName("Dummy Calc.")]
        public string DummyCalculation
        {
            get
            {
                return (dummyCalculation);
            }
        }


        // add more Calculation properties here
        // ...


        private int dummyParameter = 0;
        /// <summary>
        /// Optional! Dummy Parameter. Put all input parameters for the calculation into this
category
        /// </summary>
        [Category(PropertyCategory.Configuration)]
        [Description("Description of DummyParameter goes here. Here: not used.")]
        [DefaultValue(0)]
        [DisplayName("Dummy Parameter")]
        public int DummyParameter
        {
            get
            {
                return (dummyParameter);
            }
            set
            {
                dummyParameter = value;
            }
        }


        // add more Configuration properties here
        // ...


        #endregion

        #region Constructor
        /// <summary>
        /// Constructor
        /// </summary>
        public CustomLibFunction()
        {
            // add initialisation code if necessary
        }
        #endregion

        #region calculation function
        /// <summary>
        /// // THE calculate function. Do not change name or parameter! This function will be
searched for
        /// by reflection and called for execution, when this library is added to an analy-
sis.
        /// </summary>
        public BitArray Calculate(BitArray buf)
        {
            if (buf == null)
                return (null);

            if (buf.Count <= 0)
            {
                outbox.Length = 0;
                return (outbox);
```

```
            }

            //---------------------------------------------------------------------
            // any calculation goes here
            //---------------------------------------------------------------------

            outbox = (BitArray)buf.Clone();
            outbox.Not();

            this.dummyCalculation = outbox.Count.ToString();    // any calculation results of
interest will be reflected in the property grid

            return (outbox);    // must return the result

        }
        #endregion
    }
}
```

Source Code Template / Example (C# .NET for Matlab)

BVCustLibMatlab.cs

```
//-----------------------------------------------------------------------------
// File        : BVCustLibMatlab.cs
// Author      : Wavecom Elektronik AG
// Date        : February 2008
// Description : Template/example for a custom defined library function.
//
//               Important note:
//               In the project settings the "Assembly Name" must be
//               equal to the "Default Namespace".
//
//-----------------------------------------------------------------------------

#region using references

using System;
using System.Windows.Forms;
using System.Collections;
using System.Text;
using System.ComponentModel;
using System.Diagnostics;
using BVCustLibMatlab.Properties;

// if working with Matlab .net DLLs
using MathWorks.MATLAB.NET.Utility;
using MathWorks.MATLAB.NET.Arrays;

// reference to the Matlab dll
using MatlabFunction;

#endregion

namespace CustomLib  // Mandatory! Do not change this name.
{
    /// <summary>
    /// </summary>
    [Serializable]
    [DefaultProperty("FunctionName")] // adjust string if necessary, it points to a property
below
    public class BVCustLibMatlab
    {
        #region Constant fields

        private class PropertyCategory
        {
            // these are the categories in the BitViewTool property grid (Parameter Window)
            public const string Information = "Information";
            public const string Counter = "Counter";
            public const string Configuration = "Configuration";
            public const string Calculation = "Calculation";
```

```
            // do not change these names, they are used to distinguish the different categories
            // when any change in the parameters happens
        }

        private const int MAX_SIZE = 500000; // do not modify this value, it specifies maximum number of bits processed by BitViewTool

        #endregion

        #region Fields

        // generate a new BitArray for calculation, it will be returned to the calling function
        private BitArray outbox = new BitArray(MAX_SIZE);

        #endregion

        #region Mandatory Properties

        private string functionName = "Test CustomLib Matlab";
        /// <summary>
        /// Mandatory! Do not delete this property! The string "BVCustLibMatlab" may be modified.
        /// This will be the name of the Function in the BitViewTool History explorer and ToolBox
        /// </summary>
        [Category(PropertyCategory.Information)]
        [Description("Description of Test CustomLib Matlab goes here.\n" +
                     "Here: example of a custom defined library function. Inverts all the input bits.\n" +
                       "Inversion is performed by a Matlab function encapsulated in a .net dll assembly.")]
        [DisplayName("Function Name")]
        public string FunctionName
        {
            get
            {
                return (functionName);
            }
        }


        private string comment = string.Empty;
        /// <summary>
        /// Mandatory! Do not modify this property!
        /// </summary>
        [Category(PropertyCategory.Information)]
        [Description("My comments.")]
        [DefaultValue("")]
        public string Comment
        {
            get
            {
                return comment;
            }
            set
            {
                comment = value;
            }
        }

        private static System.Drawing.Image iconImage = Resources.matlab;
        /// <summary>
        /// If available, you can specify a custom image displayed on the function button
        /// in the BitView toolbox. The image must be 16x16 pixel. The bitmap file must be
        /// imported into the CustomLibFunction resources, so that it can be refered to it as shown above!
        /// Currently the image is the .bmp file 'CustomFunction.bmp'. It is part of this solution, it can be
        /// seen in the Solution Explorer Window and it can be modified.
        /// If there is no image available, remove all private and public fields, i.e. iconImage and IconImage,
        /// iconTransparentColor and IconTransparentColor
        /// below. A default icon image is then added by the BitViewTool main application.
        /// </summary>
        [Category(PropertyCategory.Information)]
```

```
        [Description("Icon bitmap for Toolbox Button and History Explorer")]
        [Browsable(false)]
        public static System.Drawing.Image IconImage
        {
            get
            {
                return iconImage;
            }
        }

        private     static     System.Drawing.Color     iconTransparentColor     =     Sys-
tem.Drawing.Color.FromArgb(224, 223, 227);
        /// <summary>
        /// If available, you can specify a custom image display on the function button
        /// in the toolbox. The image must be 16x16 pixel, the transparent color must
        /// be specified here
        /// </summary>
        [Category(PropertyCategory.Information)]
        [Description("Icon bitmap transparent color")]
        [Browsable(false)]
        public static System.Drawing.Color IconTransparentColor
        {
            get
            {
                return iconTransparentColor;
            }
        }

        #endregion

        #region Optional Properties

        //  Input Parameter for calculation function

        private int inpar1 = 13; // set a reasonable default value
        /// <summary>
        /// optional parameter for calculation, put this into the Configuration category
        /// </summary>
        [Category(PropertyCategory.Configuration)]
        [Description("Description of Input 1 goes here")]
        [DefaultValue(13)]   // set a reasonable default value
        [DisplayName("Input 1")]
        public int Inpar1
        {
            get
            {
                return inpar1;
            }
            set
            {
                inpar1 = value;
            }
        }


        private int inpar2 = 5; // set a reasonable default value
        /// <summary>
        /// optional parameter for calculation, put this into the Configuration category
        /// </summary>
        [Category(PropertyCategory.Configuration)]
        [Description("Description of Input 2 goes here")]
        [DefaultValue(5)]   // set a reasonable default value
        [DisplayName("Input 2")]
        public int Inpar2
        {
            get
            {
                return inpar2;
            }
            set
            {
                inpar2 = value;
            }
        }


        // add more Configuration properties here
```

```
        // ...


        //  results from calculation function

        private int outpar1 = 0;
        /// <summary>
        /// Optional result form this function! Put all results into the calculation category
        /// </summary>
        [Category(PropertyCategory.Calculation)]
        [Description("Description of Output 1 goes here.")]
        [DisplayName("Output 1")]
        public int Outpar1
        {
            get
            {
                return outpar1;
            }
        }


        private int outpar2 = 0;
        /// <summary>
        /// Optional result form this function! Put all results into the calculation category
        /// </summary>
        [Category(PropertyCategory.Calculation)]
        [Description("Description of Output 2 goes here.")]
        [DisplayName("Output 2")]
        public int Outpar2
        {
            get
            {
                return outpar2;
            }
        }

        private int outpar3 = 0;
        /// <summary>
        /// Optional result form this function! Put all results into the calculation category
        /// </summary>
        [Category(PropertyCategory.Calculation)]
        [Description("Description of Output 3 goes here.")]
        [DisplayName("Output 3")]
        public int Outpar3
        {
            get
            {
                return outpar3;
            }
        }

        private int outpar4 = 0;
        /// <summary>
        /// Optional result form this function! Put all results into the calculation category
        /// </summary>
        [Category(PropertyCategory.Calculation)]
        [Description("Description of Output 4 goes here.")]
        [DisplayName("Output 4")]
        public int Outpar4
        {
            get
            {
                return outpar4;
            }
        }


        private string outpar5 = string.Empty;
        /// <summary>
        /// Optional result form this function! Put all results into this category
        /// </summary>
        [Category(PropertyCategory.Calculation)]
        [Description("Description of Output 5 (string) goes here.")]
        [DisplayName("Output 5")]
        public string Outpar5
        {
```

```
            get
            {
                return (outpar5);
            }
        }


        // add more Calculation properties here
        // ...

        #endregion

        #region Constructor
        /// <summary>
        /// Constructor
        /// </summary>
        public BVCustLibMatlab()
        {
            // add initialisation code if necessary
        }
        #endregion

        #region calculation function
        /// <summary>
        /// // THE calculate function. Do not change name or parameter! This function will be
searched for
        /// by reflection and called for execution, when this library is added to an analy-
sis.
        /// </summary>
        public BitArray Calculate(BitArray buf)
        {
            if (buf == null)
                return (null);

            if (buf.Count <= 0)
            {
                outbox.Length = 0;
                return (outbox);
            }

            //----------------------------------------------------------------------
            // any calculation goes here
            //----------------------------------------------------------------------

            try      // catch any exception from matlab dll. If calling the matlab function
fails, we do not crash the app
            {
                // Generate data arrays for Matlab function.
                // Data input for matlab function is derived from buf, which is type of Bi-
tArray
                // Matlab has not a compatible data type, so we have to convert BitArray buf
                // into the Matlab default numeric type double 0.0 and 1.0

                // new instance of a Matlab numeric array with a size of [buf.Length,1]
                MWNumericArray data = new MWNumericArray(MWArrayComplexity.Real, MWNumeric-
Type.Double, new int[] { buf.Length, 1 });

                // Initialize data from BitArray buf
                // Matlab index range is from 1...n, in C# it is from 0 ... (n-1)
                for (int idx = 1; idx <= buf.Length; idx++)
                {
                    data[idx, 1] = buf[idx - 1] ? 1.0 : 0.0;
                }

                // data output from matlab function is an Array of arguments
                MWArray[] argsOut = null;

                // prepare for calling the Matlab function, create an instance of the Matlab
function class
                // generated by the Matlab deploytool, use Intellisense in visual studio,
then you see
                // the list of classes found up to now in this project. Our example is
                // Matlab function called 'MatlabFunction' in the .m File with the same name.
                // Here is the complete listing of the Matlab .m File:

                /*----------------------------------------------------------------------------
--------------------
```

```
                function [y, outpar1, outpar2, outpar3, outpar4, outpar5] = MatlabFunction(
x, inpar1, inpar2 )
                %
                %--------------------------------------------------------------------------
                % Example of a function to be called from .Net environment, especially from
                % a BitViewTool CustomLib function. This function declares different input
                % and output parameters, to demonstrate how to access these parameters from
                % a .Net environment.
                %
                % Inputs:
                % -------
                %
                % x     :  should be a [n,1] input array containing the bit stream from the
                %          BitViewTool, the expected values must be 0 and 1
                % inpar1, inpar2:  these are parameters to control the function's behaviour
                %
                % Outputs:
                % --------
                %
                % y     :  should be a [k,1] output array containing the bit stream as an
                %          output from this function, the type of y should be double or logi-
cal.
                % The values must be 0.0 and 1.0 if double.
                % outpar1, outpar2, outpar3, outpat4, outpar5 : these are additional calcula-
tion
                % results from this function of type scalar or string depending on the
                % function's behaviour. outpar5 is of type char array
                %
                %
                %--------------------------------------------------------------------------
                % After this function is debugged and tested, the Matlab deploytool has to be
                % started with 'deploytool' in the Command Window
                % In the deploytool create a new .net project, project type is .NET
                % Component, enter a Component name, in the project settings under .NET set
the
                % the Microsoft Framework to Version 2.0, the Assembly type to private.
                % Then push the Build Project button in the Deployment toolbar.
                % When finished copy the .dll and .ctf files from the function's
                % project\distrib directory to the BitViewTool customlib directory
                %                    C:\Documents         and         Settings\All         Us-
ers\Documents\WAVECOM\BitViewTool\CustomLib
                %
                % Next step is to create a new WAVECOM CustomLib function from the template
                % found in the new project wizzard in Visual Studio.
                % Add a the reference in the Solution Explorer to the .dll just copied to to
                % CustomLib directory. Add a reference in the C# CustomLib source under the
                % 'using' region, the namespace to be used is the same used in the solution
                % explorer.
                %
                %--------------------------------------------------------------------------

                outpar1 = nargin;    % for example : outpar1 returns number of input function
arguments
                [m, n] = size( x );    % for example : outpar2 returns length of input array
x, i.e. number of rows
                outpar2 = m;
                outpar3 = n;        % number of columns, should be 1 in our example

                outpar4 = inpar1 * inpar2;  % example of calculation for outpar4

                y = ~x;              % this is the only calculation for the input data, outpt
y is inverse(x)
                                    % and converts y to type 'logical' !!!!
                if (n ~= 1)
                    outpar5 = 'Function error: input data dimension [m,n] with n ~= 1';
                else
                    outpar5 = 'Function called successfully';
                end
                ------------------------------------------------------------------------------
--------
               */


                // we can now see 3 Input parameters and 6 output parameters
                // x is the input data column vector, type is double
                // y is the returned column vector, type can be double or logical,
```

```
                // all other parameters are scalars, expected
                // to be of type double, outpar5 is of type string
                // Use Intellisense to see all available classes
                // the matlab deploytool generates a class by using the Matlab function name
                // with 'class' appended:

                MatlabFunctionclass MatlabFunc = new MatlabFunctionclass(); // new instance
of Matlab function

                // now call this function, with 6 output parameters : y and outpar 1...5
                argsOut = MatlabFunc.MatlabFunction(6, data, (double)Inpar1, (double)Inpar2);

                // now get all results from argsOut, its an array of numeric/char arrays
                // the first entry is our bit array result, can be of type numeric(double) or
logical,
                // depending on the matlab function internal code

                if  ((argsOut[0].IsNumericArray)  &&  (argsOut[0].NumberofDimensions  ==  2))
// if numeric and number of dimension exactly 2
                {
                    MWNumericArray numericBits = (MWNumericArray)argsOut[0];      // cast to
MWNumericArray
                    MWNumericType numericType = numericBits.NumericType;          // get the
numeric type, can be double, float, int, etc..., this has to be checked
                    Array  numericBitsArray  =  numericBits.ToVector(MWArrayComponent.Real);
// must be anything but not complex !!!
                    int length;

                    switch (numericType)    // check types, we expect double
                    {
                        case MWNumericType.Double:
                            double[] doubleArray = (double[])numericBitsArray;   // cast to
double array
                            length = doubleArray.Length;
                            if (length > MAX_SIZE)
                            {
                                length = MAX_SIZE;
                            }
                            // convert now result to type of BitArray
                            for (int i = 0; i < length; ++i)
                            {
                                outbox[i] = (doubleArray[i] == 0.0) ? false : true;
                            }
                            outbox.Length = length; // set length explicitely, otherwise we
have MAX_SIZE !!
                            break;

                        case MWNumericType.Int32:
                            Int32[] int32Array = (Int32[])numericBitsArray;  // cast to int32
array
                            length = int32Array.Length;
                            if (length > MAX_SIZE)
                            {
                                length = MAX_SIZE;
                            }
                            // convert now result to type of BitArray
                            for (int i = 0; i < length; ++i)
                            {
                                outbox[i] = (int32Array[i] == 0) ? false : true;
                            }
                            outbox.Length = length; // set length explicitely, otherwise we
have MAX_SIZE !!
                            break;

                        case MWNumericType.Int16:
                            Int16[] int16Array = (Int16[])numericBitsArray;  // cast to int16
array
                            length = int16Array.Length;
                            if (length > MAX_SIZE)
                            {
                                length = MAX_SIZE;
                            }
                            // convert now result to type of BitArray
                            for (int i = 0; i < length; ++i)
                            {
                                outbox[i] = (int16Array[i] == 0) ? false : true;
                            }
```

```
                            outbox.Length = length; // set length explicitely, otherwise we
have MAX_SIZE !!
                            break;

                        case MWNumericType.Int8:
                            Byte[] byteArray = (Byte[])numericBitsArray;   // cast to int16
array
                            length = byteArray.Length;
                            if (length > MAX_SIZE)
                            {
                                length = MAX_SIZE;
                            }
                            // convert now result to type of BitArray
                            for (int i = 0; i < length; ++i)
                            {
                                outbox[i] = (byteArray[i] == 0) ? false : true;
                            }
                            outbox.Length = length; // set length explicitely, otherwise we
have MAX_SIZE !!
                            break;


                        default:    // other numeric types not supported
                            throw new ApplicationException("Bad type returned from " + Func-
tionName);
                    }
                }
                else if ((argsOut[0].IsLogicalArray) && (argsOut[0].NumberofDimensions == 2))
// if logical and number of dimension exactly 2
                {
                    MWLogicalArray logicalBits = (MWLogicalArray)argsOut[0];          // cast
to MWLogicalArray

                    bool[] boolOutputArray;
                    boolOutputArray = logicalBits.ToVector();        // convert all to a bool
array in .net
                    outbox = new BitArray(boolOutputArray);          // then initialize a Bi-
tArray for use with BitViewTool
                }
                else
                {
                    throw new ApplicationException("Bad type returned from " + FunctionName);
                }

                // now get other results, we do not check types again, but you should do that
                outpar1 = (int)(MWNumericArray)argsOut[1];
                outpar2 = (int)(MWNumericArray)argsOut[2];
                outpar3 = (int)(MWNumericArray)argsOut[3];
                outpar4 = (int)(MWNumericArray)argsOut[4];
                outpar5 = ((MWCharArray)argsOut[5]).ToString();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message.ToString(), "Errors in " + FunctionName);
                outbox.Length = 0;
            }

            return (outbox);

        }
        #endregion
    }
}
```

Source Code Template / Example (Matlab)

MatlabFunction.m

```
function [y, outpar1, outpar2, outpar3, outpar4, outpar5] = MatlabFunction( x, inpar1, inpar2
)
%
%--------------------------------------------------------------------------
% Example of a function to be called from .Net environment, especially from
```

```
% a BitViewTool CustomLib function. This function declares different input
% and output parameters, to demonstrate how to access these parameters from
% a .Net environment.
%
% Inputs:
% -------
%
% x     :  should be a [n,1] input array containing the bit stream from the
%          BitViewTool, the expected values must be 0 and 1
% inpar1, inpar2:  these are parameters to control the function's behaviour
%
% Outputs:
% --------
%
% y     :  should be a [k,1] output array containing the bit stream as an
%          output from this function, the type of y should be double or logical.
% The values must be 0.0 and 1.0 if double.
% outpar1, outpar2, outpar3, outpat4, outpar5 : these are additional calculation
% results from this function of type scalar or string depending on the
% function's behaviour. outpar5 is of type char array
%
%-------------------------------------------------------------------------
% After this function is debugged and tested, the Matlab deploytool has to be
% started with 'deploytool' in the Command Window
% In the deploytool create a new .net project, project type is .NET
% Component, enter a Component name, in the project settings under .NET set the
% the Microsoft Framework to Version 2.0, the Assembly type to private.
% Then push the Build Project button in the Deployment toolbar.
% When finished copy the .dll and .ctf files from the function's
% project\distrib directory to the BitViewTool customlib directory
%  C:\Documents and Settings\All Users\Documents\WAVECOM\BitViewTool\CustomLib
%
% Next step is to create a new WAVECOM CustomLib function from the template
% found in the new project wizzard in Visual Studio.
% Add a the reference in the Solution Explorer to the .dll just copied to to
% CustomLib directory. Add a reference in the C# CustomLib source under the
% 'using' region, the namespace to be used is the same used in the solution
% explorer.
%
%-------------------------------------------------------------------------

outpar1 = nargin;   % for example : outpar1 returns number of input function arguments
[m, n] = size( x );   % for example : outpar2 returns length of input array x, i.e. number of
rows
outpar2 = m;
outpar3 = n;        % number of columns, should be 1 in our example

outpar4 = inpar1 * inpar2;  % example of calculation for outpar4

y = ~x;             % this is the only calculation for the input data, output y is inverse(x)
                % this converts y to type 'logical' !!!!

if (n ~= 1)
    outpar5 = 'Function error: input data dimension [m,n] with n ~= 1';
else
    outpar5 = 'Function called successfully';
end
```

Glossary of Terms

ANSI

An acronym for the American National Standards Institute, an organization that sets standards for a variety of programming languages and systems.

ASCII

An acronym for American Standard Code for Information Interchange, pronounced "ASK-ee." It is a code in which the numbers from 0 to 127 stand for letters, numbers, punctuation marks and other characters. ASCII code is standardized to facilitate transmitting text between computers or between a computer and a peripheral device.

BCH

A BCH (Bose, Ray-Chaudhuri, Hocquenghem) code is an error-correcting code. It is a multilevel, cyclic, error-correcting, variable-length digital code used to correct mutiple random error patterns.

Convolutional code

A type of channel coding that adds patterns of redundancy to the data in order to improve the signal-to-noise ratio (SNR) for more accurate decoding at the receiving end. The Viterbi algorithm is used to decode a particular type of convolutional code

CRC

CRC (Cyclical Redundancy Checking) is an error checking technique used to ensure the accuracy of transmitting digital data. The transmitted messages are divided into predetermined lengths which, used as dividends, are divided by a fixed divisor. The remainder of the calculation is appended onto and sent with the message. At the receiving end, the computer recalculates the remainder. If it does not match the transmitted remainder, an error is detected.

HEX

In mathematics and computer science, hexadecimal, or simply hex, is a numeral system with a radix or base of 16 usually written using the symbols 0–9 and A–F or a–f.

HDLC

HDLC (High-level Data Link Control) is a group of protocols for transmitting synchronous data packets between point-to-point nodes. In HDLC, data is organized into a frame. HDLC uses zero insertion/deletion process (bit stuffing) to ensure that the bit pattern of the delimiter flag does not occur in the fields between flags.

Matlab

MATLAB is a numerical computing environment and programming language. Created by The MathWorks, MATLAB allows easy matrix manipulation, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs in other languages. Although it specializes in numerical computing, an optional toolbox interfaces with the Maple symbolic engine, allowing it to be part of a full computer algebra system.

Index

A

Adding a Custom Function 40
Analysis Sets 6
Analysis Tools 33
AND / OR / XOR / NOT 21
ANSI 55
Arabic Baghdad-70 31
Arabic Baghdad-80 (ATU-80) 31
ARQ-E 27
ASCII 33, 55
Autocorrelation 34

B

BAUER 28
BCH 55
BCH-Decoding 24
Binary Modulation 15
Bi-Phase-L (Manchester) 16
Bi-Phase-M 16
Bi-Phase-S 17
Bit Manipulation 18
Bit Statistic 33
Bit Stream Processing 5
Bit Sync Analysis 35
Bulgarian 32
BVCustLibMatlab.cs 46

C

Channel Decoding (Protocol) 27
Constraints 41
Convolutional code 55
CRC 55
CRC & Polynomial 24
CRC (1..32) 24
CRC-10 25
CRC-12 26
CRC-16 26
CRC-32 27
CRC-8 25
CRC-CCITT 26
Custom Library 38
CustomLibFunction.cs 43
Cutting 22
Cyrillic 30

D

Danish-Norwegian 32
DBi-Phase-M 17
DBi-Phase-S 17
Decoding/Equalizer 22
De-Interleave Bit Block 20
De-Interleaving Stream 20
De-Puncturing 23
De-Stuffing (HDLC) 18
Difference-Decoding 23

E

Export Text Data 15
Extraction 21

F

FEC-A 28
French 32
Function Library 14

G

General 1
German 32
Getting Started 3
Glossary of Terms 55

H

HDLC 55
Hebrew 31
HEX 55
History Explorer Window 9
HNG-FEC 28

I

Import Binary Data 14
Import Hex Data 14
Import IAS Bitstream 14
Import Text Data 14
Important Notes 41
Installation 2
Introduction 1
ITA-3 (M.342) 29
ITA-5 29

L

Latin 30
Layout Settings 10
Limitations 1

M

Matlab 55
MatlabFunction.m 53
Menu 5
Mirroring 18

N

NRZ-I 15
NRZ-M 15
NRZ-S 16

O

Open Issues 1

P

Paths 3
Polarity 19
Preamble 15
Preferences 9
Program Start 3
Properties Window 8
PSK-31 30

R

Reports 7
Requirements 1
Revisions 1
Rotation 19
RUM-FEC 29

S

Setup 2
Shift 19
Signal Duration 34
SITOR 28
Source Code Template / Example (C# .NET for Matlab) 46
Source Code Template / Example (C# .NET) 43
Source Code Template / Example (Matlab) 53
Source Decoding (Alphabet) 30
Source/Sink 14
Steps to Write a Custom Function in C# .NET 41
Steps to Write a Custom Function with Matlab 42
Swedish 32
Synchronization 15

T

Tass-Cyrillic 31
Third-shift Cyrillic 31
Third-shift Greek 30
Toolbox 9

U

UNICODE 33
US 32
UTF-7 33
UTF-8 33

V

Viterbi-Decoding 22

W

Want to Roll Your Own functions? 38