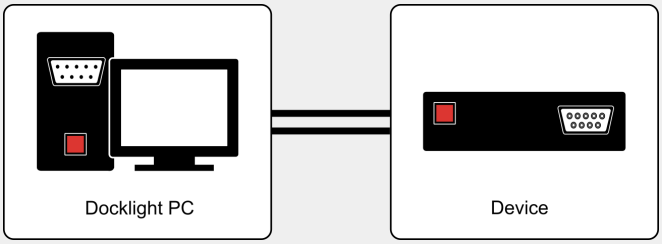
Docklight V2.4 User Manual 02/2023



Copyright

1 Copyright

Trademarks

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

Microsoft and *Windows* are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Disclaimer

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Contact

E-Mail Support: support@docklight.de
www.docklight.de

Flachmann und Heggelbacher GmbH & Co. KG
Waldkirchbogen 27
D-82061 Neuried
Germany
www.fuh-edv.de

Kickdrive Software Solutions e.K.
Robert-Bosch-Str. 5
D-88677 Markdorf
Germany
www.kickdrive.de

Introduction

2 Introduction

2.1 Docklight - Overview

Docklight is a testing, analysis, and simulation tool for serial communication protocols (RS232, RS485/422 and others). It allows you to monitor communications between two serial devices or to test the serial communication of a single device. Docklight is easy to use and works on almost any standard PC running *Windows 11*, *Windows 10, Windows 8,* or *Windows 7*.

Docklight's key functions include

- **simulating serial protocols** - Docklight can send out user-defined sequences according to the protocol used and it can react to incoming sequences. This makes it possible to simulate the behavior of a serial communication device, which is particularly useful for generating test conditions that are hard to reproduce with the original device (e.g. problem conditions).
- **logging RS232 data** - All serial communication data can be logged using two different file formats. Use plain text format for fast logging and storing huge amounts of data. An HTML file format, with styled text, lets you easily distinguish between incoming and outgoing data or additional information. Docklight can also log any binary data stream including ASCII 0 <NUL> bytes and other control characters.
- **detecting specific data sequences** - In many test cases, you will need to check for a specific sequence within the RS232 data that indicates a problem condition. Docklight manages a list of such data sequences for you and can perform user-defined actions after detecting a sequence, e.g. taking a snapshot of all communication data before and after the error message was received.
- **responding to incoming data** - Docklight lets you specify user-defined answers to the different communication sequences received. This allows you to build a basic simulator for your serial device within a few minutes. It can also help you to trace a certain error by sending out a diagnostics command after receiving the error message.

Docklight will work with the COM communication ports provided by your operating system. Physically, these ports will be RS232 SUB D9 interfaces in many cases. However, it is also possible to use Docklight for other communication standards such as RS485 and RS422, which have a different electrical design to RS232 but follow the RS232 communication mechanism.

Docklight has also been successfully tested with many popular USB-to-Serial converters, Bluetooth serial ports, GPS receivers, virtual null modems, Arduino, MicroPython/pyboard or other Embedded/UART boards that add a COM port in Windows.

For RS232 full-duplex monitoring applications, we recommend our Docklight Tap USB accessory or our Docklight Monitoring Cable.

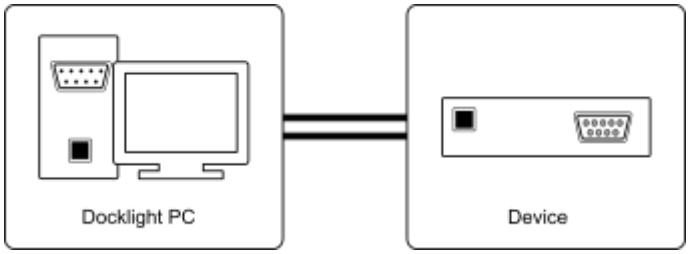
This manual only refers to RS232 serial connections in detail, since this is the basis for other serial connections mentioned above.

TIP: For getting started, have a look at the Docklight sample projects, which demonstrate some of the basic Docklight functions.

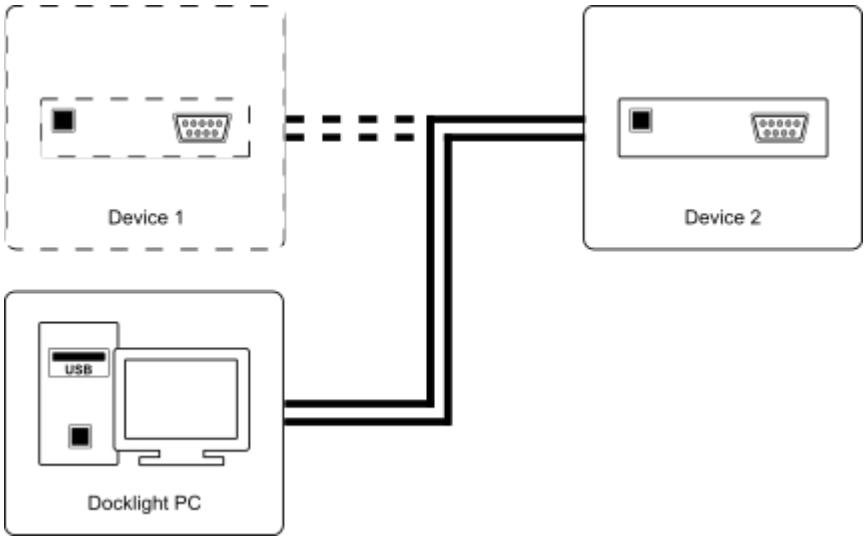
2.2 Typical Applications

Docklight is the ideal tool to support your development and testing process for serial communication devices. Docklight may be used to

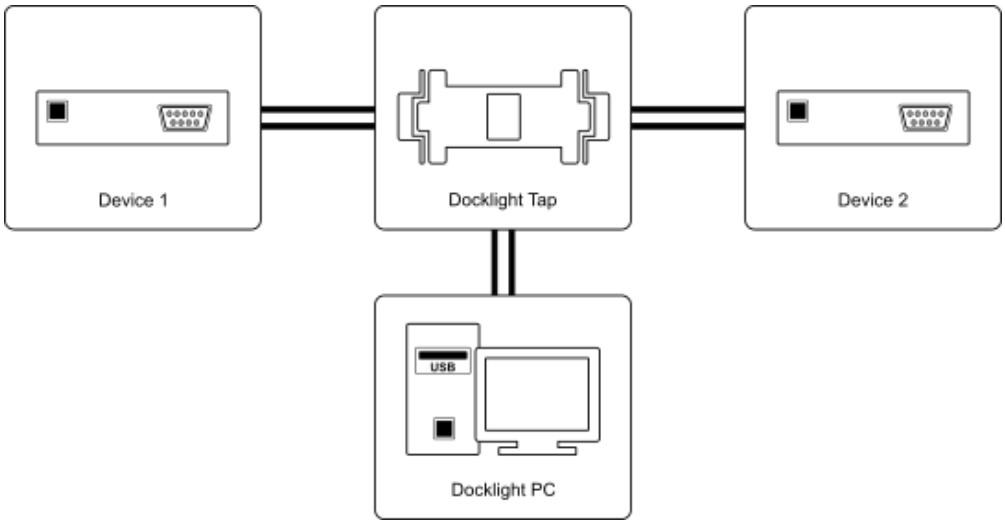
- Test the functionality or the protocol implementation of a serial device.
  You may define control sequences recognized by your device, send them, log and analyze the responses and test the device reaction.



- Simulate a serial device.
  Although rare, the possibility of a hardware fault must be considered in most systems. Imagine you have a device that sends an error message in the case of a hardware fault. A second device should receive this error message and perform some kind of reaction. Using Docklight you can easily simulate the error message to be sent and test the second device's reaction.



- Monitor the communication between two devices.
  Insert Docklight into the communication link between two serial devices. Monitor and log the serial communication in both directions. Detect faulty communication sequences or special error conditions within the monitored communication. Take a snapshot of the communication when such an error condition has occurred.



2.3 System Requirements

Operating system

- *Windows 11, Windows 10, Windows 10 x64, Windows 8, Windows 8 x64, Windows 7, Windows 7 x64.*

Additional requirements

- For RS232 testing or simulation: Minimum one COM port available. Two COM ports for monitoring communication between two serial devices.
- For low-latency monitoring using Docklight Tap, Docklight Tap Pro or Docklight Tap 485: One USB port.

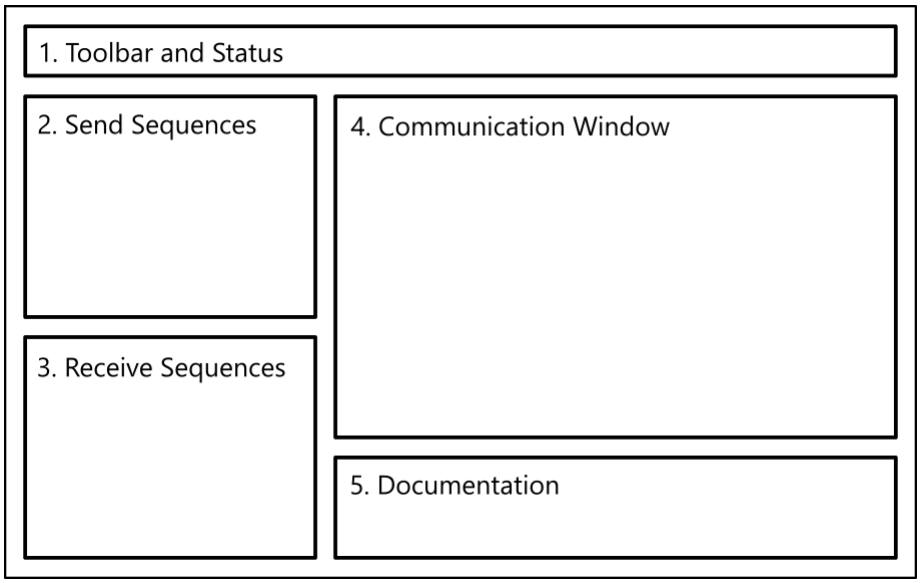
Additional cables or software drivers may be required for connecting the equipment to be tested. See the sections on Docklight Tap, Docklight Monitoring Cable RS232 SUB D9, Standard RS232 Cables and virtual null modem drivers.

User Interface

3 User Interface

3.1 Main Window

Docklight's main window is divided into four sections:



1. Toolbar and Status
You can select all main Docklight functions from the Toolbar. The status line below shows additional information about the communication status and the current settings.

2. Send Sequences
Define, edit and manage your Send Sequences here. Use the arrow symbol or the Space key to send out the selected sequence. Double click on the blank field at the end of a list to create a new sequence. A context menu (right mouse button) is available to cut, copy or paste entire Send Sequences to/from the clipboard. See Editing and Managing Sequences and Dialog: Edit Send Sequence for more information.

3. Receive Sequences
Define, edit and manage your Receive Sequences here. Double click on the blank field at the end of a list to create a new sequence. The Receive Sequence list supports the same reordering and clipboard operations as the Send Sequence list. You can also copy a Send Sequence to the clipboard and paste it into the Receive Sequence list. See Editing and Managing Sequences and Dialog: Edit Receive Sequence for more information.

You can reorder the sequence lists using drag&drop: First, allow reordering the list by clicking on the small lock icon in the top left corner. When unlocked, the list can be changed by dragging a sequence to a new position with the left mouse button pressed.

By clicking the **|<** mark you can minimize the Send/Receive Sequences area.

4. Communication Window
Displays the outgoing and incoming communication of the serial data connection. Various display options are available for communication data, including ASCII / HEX / Decimal / Binary display, timestamps, and highlighting (see Options). If serial communication is stopped, all data from the communications window may be copied to the clipboard or printed. You may also search for specific sequences using the Find

Sequence function. See How Serial Data is Processed and Displayed for more information.

5. Project and Sequence Documentation
Type in additional comments concerning your project, or a specific Send Sequence / Receive Sequence. Docklight presents sequence-specific documentation when you choose a Send Sequence or Receive Sequence from the list (2. and 3.). Docklight switches to the main project documentation when you click on the empty bottom line of the sequence list, or when you click inside the Communication Window (4.).

To avoid accidental editing, the Documentation Area is locked by default and you need to enable editing by clicking on the small lock icon above it. When unlocked, you can edit/copy/paste/delete its contents freely.

By clicking the **v** mark on the right side you can minimize the documentation area.

3.2 Clipboard - Cut, Copy & Paste

Docklight supports Cut/Copy/Paste operations. Clipboard operations are available in the
- Main Window - Send Sequences
- Main Window - Receive Sequences
- Main Window - Communication
- Main Window - Documentation
- Main Window - Script Editor (Docklight Scripting only)
- Dialog: Edit Send Sequence
- Dialog: Edit Receive Sequence
- Dialog: Find Sequence
- Dialog: Send Sequence Parameter
- Documentation Area
- Keyboard Console

You can cut a serial data sequence from the communication window and create a new Send or Receive Sequence by pasting it into the appropriate list. Or edit a Send Sequence, copy a part of this sequence to the clipboard and create a new Receive Sequence from it by pasting it into the Receive Sequence window.

TIP: Use the **right mouse button** context menu for Cut/Copy/Paste operations or the related Keyboard Hotkey.

3.3 Documentation Area

Docklight offers documentation areas in the lower right part of the main window and in the Edit Send Sequence or Edit Receive Sequence dialogs.

You can use these areas to write down additional notes concerning your Docklight application. E.g., how to use the Send / Receive Sequences and sequence parameters, or notes on additional test equipment, etc.

The documentation contents are stored and loaded along with all other Docklight project settings (see saving and loading your project data).

TIP: The documentation areas are simple text boxes without formatting menus or tools. For formatted documentation including pictures and tables, you can prepare your documentation in *Windows WordPad* or *Microsoft Word* and use copy&paste to add it to the Docklight documentation area.

Features and Functions

4 Features and Functions

4.1 How Serial Data Is Processed and Displayed

Docklight handles all serial data in an 8 bit-oriented way. Every sequence of serial data consists of one or more 8 bit characters. Docklight allows you to

- display the serial data in either ASCII, HEX, Decimal or Binary format
- copy serial data to the clipboard and paste it into a standard text file or a formatted *Microsoft® Word* document, or create a Send / Receive Sequence using the data.
- print out serial data, user comments and other information

Docklight's communication window shows the current communication on the selected serial port(s). Docklight distinguishes between two communication channels (channel 1 and channel 2), which represent the incoming and outgoing data in Send/Receive Mode or the two communication channels being observed in Monitoring Mode. Channel 1 and channel 2 data are displayed using different colors or fonts, and the communication data may be printed or stored as a log file in plain text or HTML format.

Besides the serial data, Docklight inserts date/time stamps into the communication display. By default, a date/time stamp is inserted every time the data flow direction switches between channel 1 and channel 2, or before a new Send Sequence is transmitted. There are several options available for inserting additional time stamps. This is especially useful when monitoring a half-duplex line with only one communication channel. See Options --> Date/Time Stamps

Docklight is able to process serial data streams containing any ASCII code 0 - 255 decimal. Since there are non-printing control characters (ASCII code < 32) and different encodings for ASCII code > 127, not all of these characters can be displayed in the ASCII text window. Nonetheless, all characters will be processed properly by Docklight and can be displayed in HEX, Decimal or Binary format. Docklight will process the serial data on any language version of the *Windows* operating system in the same way, although the ASCII display might be different. For control characters (ASCII code < 32), an additional display option is available to display their text equivalent in the communication window. See Options dialog and Appendix, ASCII Character Set Tables.

Docklight allows you to suppress all original serial data, if you are running a test where you do not need to see the actual data, but only the additional evaluations generated using Receive Sequences. See the Project Settings for Communication Filter.

4.2 Editing and Managing Sequences

A Docklight project mainly consists of user-defined sequences. These may be either Send Sequences, which may be transmitted by Docklight itself, or Receive Sequences, which are used to detect a special message within the incoming serial data.

Sequences are defined using the Edit Send Sequence or Edit Receive Sequence dialog window. This dialog window is opened
1. by choosing **Edit** from the context menu available using the **right mouse button**.
2. by double-clicking on an existing sequence or pressing **Ctrl + E** with the Send Sequence or Receive Sequence list selected.
3. when creating a new sequence by double-clicking on the blank field at the end of a list (or pressing **Ctrl + E**).
4. when pasting a new sequence into the sequence list.

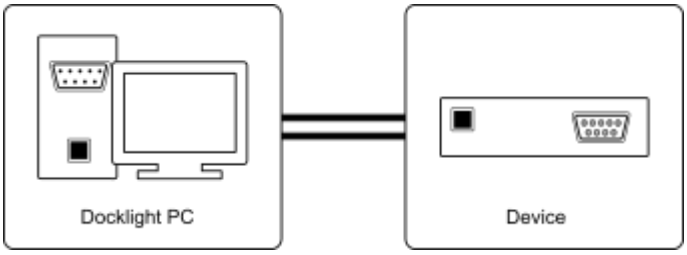
Docklight supports the use of wildcards (e.g. wildcard "?" as a placeholder for one arbitrary character) within Receive Sequences and Send Sequences. See the sections

sending commands with parameters and checking for sequences with random characters for details and examples.

Working with Docklight

5 Working with Docklight

5.1 Testing a Serial Device or a Protocol Implementation



Preconditions

- You need the specification of the protocol to test, e.g. in written form.
- The serial device to test should be connected to one of the PC's COM ports. See section Standard RS232 Cables for details on how to connect two serial devices.
- The serial device must be ready to operate.

Performing the test

A) Creating a new project
Create a new Docklight project by selecting the menu **File >** **New Project**

B) Setting the Communication Options
1. Choose the menu **Tools >** **Project Settings...**
2. Choose communication mode **Send/Receive**
3. At **Send/Receive on comm. channel**, set the COM Port where your serial device is connected.
4. Set the baud rate and all other **COM Port Settings** required.
5. Confirm the settings and close the dialog by clicking the **OK** button.

C) Defining the Send Sequences to be used
You will probably test your serial device by sending specific sequences, according to the protocol used by the device, and observe the device's reaction. Perform the following steps to create your list of sequences:

1. Double click on the last line of the Send Sequences table. The Edit Send Sequence dialog is displayed (see also Editing and Managing Sequences).
2. Enter a **Name** for the sequence. The sequence name should be unique for every Send Sequence defined.
3. Enter the **Sequence** itself. You may enter the sequence either in ASCII, HEX, Decimal or Binary format. Switching between the different formats is possible at any time using the **Edit Mode** radio buttons.
4. After clicking the **OK** button the new sequence will be added to the Send Sequence lists.

Repeat steps 1 - 4 to define the other Send Sequences needed to perform your test.

D) Defining the Receive Sequences used
If you want Docklight to react when receiving specific sequences, you have to define a list of Receive Sequences.

1. Double click on the last line of the Receive Sequences table. The dialog Edit Receive Sequence is displayed. The dialog consist of three parts: **Name** field, **Sequence** field, and **Action** field.
2. Edit the **Name** and **Sequence** fields.
3. Specify an Action to perform after the sequence has been received by Docklight. There are four types of actions available:
   Answer - After receiving the sequence, transmit one of the Send Sequences.
   Comment - After receiving the sequence, insert a user-defined comment into the communication window (and log file, if available).
   Trigger - This is an advanced feature described in Catching a specific sequence...
   Stop - After receiving the sequence, Docklight stops communications.
4. Click the **OK** button to add the new sequence to the list.

Repeat steps 1 - 4 to define the other Receive Sequences you need to perform your test.

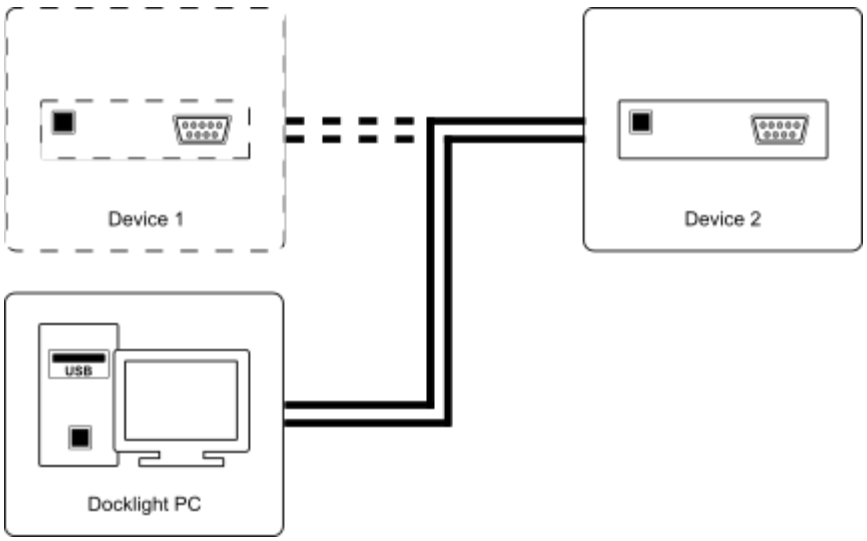
E) Storing the project
Before running the actual test, it is recommended that the communication settings and sequences defined be stored. This is done using the menu **File >** **Save Project**.

F) Running the test
Start Docklight by choosing **Run >** **Start Communication**.

Docklight will open a serial connection according to the parameters specified. It will then display all incoming and outgoing communication in the communication window. Use the ----> **Send** button to send one of the defined sequences to the serial device. The on-screen display of all data transfer allows you to check the device's behavior. All protocol information can be logged in a text file for further analysis. Please see section Logging and analyzing a test.

TIP: Using the Documentation Area , you can easily take additional notes, or copy & paste parts of the communication log for further documentation.

5.2 Simulating a Serial Device



Preconditions

- You need the specification of the behavior of the serial device you want to simulate, e.g. what kind of information is sent back after receiving a certain command.
- A second device is connected to a PC COM port, which will communicate with your simulator.

This second device and its behavior is the actual object of interest. An example could be a device that periodically checks the status of an UPS (Uninterruptible Power Supply) using a serial communication protocol. You could use Docklight to simulate basic UPS behavior and certain UPS problem cases. This is very useful when testing the other device, because it can be quite difficult to reproduce an alarm condition (like a bad battery) at the real UPS.

NOTE: The second device may also be a second software application. It is possible to run both Docklight and the software application on the same PC. Simply use a different COM port for each of the two applications and connect the two COM ports using a RS232 null modem cable. You can also use a virtual null modem for this purpose.

Performing the test

A) Creating a new project
Create a new Docklight project by selecting the menu **File >** **New Project**

B) Setting the Communication Options
1. Choose the menu **Tools >** **Project Settings...**
2. Choose communication mode **Send/Receive**
3. At **Send/Receive on comm. channel**, set the COM Port where your serial device is connected.
4. Set the baud rate and all other **COM Port Settings** required.
5. Confirm the settings and close the dialog by clicking the **OK** button.

C) Defining the Send Sequences used
Define all the responses of your simulator. Think of responses when the simulated device is in normal conditions, as well as responses when in fault condition. In the UPS example mentioned above, a battery failure would be such a problem case that is hard to reproduce with the original equipment. To test how other equipment reacts to a battery failure, define the appropriate response sequence your UPS would send in this case.

NOTE: See Testing a serial device... to learn how to define Send Sequences.

D) Defining the Receive Sequences used
In most cases, your simulated device will not send unrequested data, but will be polled from the other device. The other device will use a set of predefined command sequences to request different types of information. Define the command sequences that must be interpreted by your simulator here.

For every command sequence defined, specify **Answer** as an action. Choose one of the sequences defined in C). If you want to use two or more alternative response sequences, make several copies of the same Receive Sequence, give them a different name (e.g. "status cmd - answer ok", "status cmd - answer battery failure", "status cmd - answer mains failure") and assign different Send Sequences as an action. In the example, you would have three elements in the Receive Sequences list that would respond to the same command with three different answers. During the test you may decide which answer should be sent by checking or unchecking the list elements using the **Active** column.

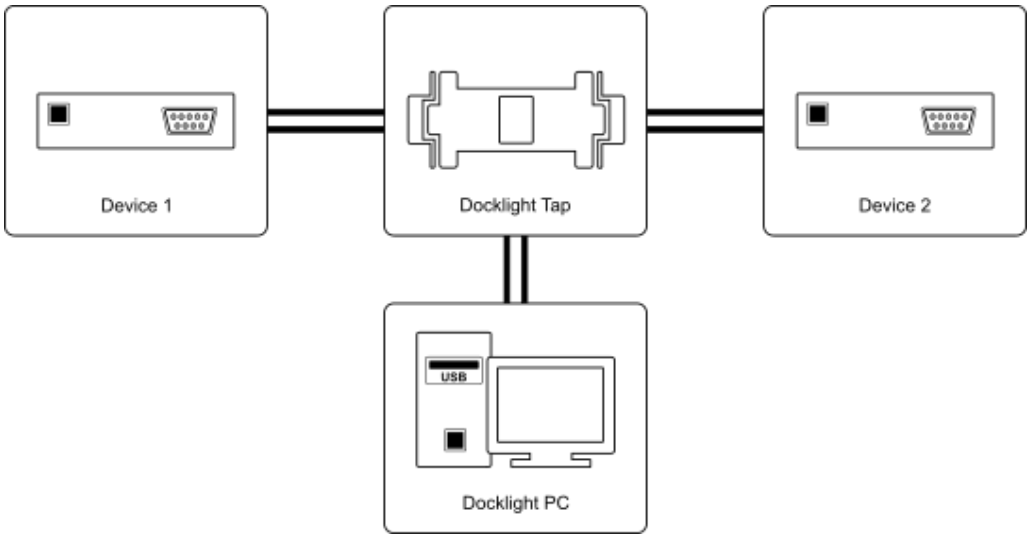
E) Storing the project
Before running the actual test, it is recommended that the communication settings and sequences defined be stored. This is done using the menu **File >** **Save Project**.

F) Running the test
Start Docklight by choosing **Run >** **Start Communication**.

Docklight will now respond to all commands received from the connected serial device. The on-screen data transfer display allows you to monitor the communications flow. All protocol information can be logged to a text file for further analysis. See section Logging and analyzing a test.

TIP: Using the Documentation Area, you can easily take additional notes, or copy & paste parts of the communication log for further documentation.

5.3 Monitoring Serial Communications Between Two Devices



Preconditions

- A Docklight Monitoring Cable, Docklight Tap, or Docklight Tap Pro/485 is required to tap the RS232 TX signals of both serial devices and feed them into Docklight, while not interfering with the communications between the devices.
- For a Docklight Monitoring Cable setup, two COM ports must be available on your PC for monitoring. Each port will receive the data from one of the serial devices being monitored.
- Device 1 and Device 2 must be ready to operate.

Performing the test

A) Creating a new project
Create a new Docklight project by selecting the menu **File >** **New Project**

B) Setting the Communication Options

1. Choose the menu **Tools >** **Project Settings...**

2. Choose communication mode **Monitoring**

Alternative I - Using Docklight Monitoring Cable:

3. At **Receive Channel 1**, set the COM Port where the monitoring signal from serial device 1 is received. At **Receive Channel 2**, set the COM port for the second device.

   NOTE: In Docklight Monitoring Mode, all received data from one COM port is re-sent on the TX channel of the opposite COM port ("Data Forwarding"). This does not have any effect on Docklight Monitoring Cable setups, since the TX signal is not connected. But it can be useful for special applications where you need to route the serial data traffic through Docklight using standard RS232 cabling. If you require a pure passive monitoring behavior where no TX data appears, you can disable the "Data Forwarding" using the menu **Tools >** Expert Options...

Alternative II - Using Docklight Tap

3. At **Receive Channel 1**, open the dropdown list, scroll down to the **-- USB Taps --** section and choose the first Tap port, e.g. **TAP0**. At **Receive Channel 2**, the second tap port (e.g. **TAP1**) is selected automatically.

Alternative III - Using Docklight Tap Pro / Docklight Tap 485:

3. At **Receive Channel 1**, open the dropdown list, scroll down to the **-- USB Taps --** section and choose the first VTP Tap port, e.g. **VTP0**. At **Receive Channel 2**, the second VTP tap port (e.g. **VTP1**) is selected automatically.

4. Set the baud rate and all other communication parameters for the protocol being used.

   NOTE: Make sure your PC's serial interfaces port works properly at the baud rate and for the communication settings used by Device 1 and Device 2. If Device 1 and 2 use a high-speed data transfer protocol, the PC's serial interfaces and the Docklight software itself might be too slow to receive all data properly.

5. Confirm the settings and close the dialog by clicking the **OK** button.

C) Defining the Receive Sequences used
Define Receive Sequences, which should be marked in the test protocol or trigger an action within Docklight. Docklight checks for Receive Sequence on both monitoring channels, i.e. it does not matter whether the sequences come from serial device 1 or serial device 2.

NOTE: Since a special monitoring cable is used for this test, all communication between serial device 1 and serial device 2 will remain unbiased and no additional delays will be introduced by Docklight itself. This is particularly important when using Docklight for tracking down timing problems. This means, however, that there is no way to influence the serial communication between the two devices. While communication mode **Monitoring** is selected, it is not possible to use Send Sequences.

D) Storing the project
Before running the actual test, it is recommended to store the communication settings and sequences defined. This is done using the menu **File >** **Save Project**.

E) Running the test
Start Docklight by choosing **Run >** **Start Communication**, then activate the serial devices 1 and 2 and perform a test run. Docklight will display all communication between serial device 1 and serial device 2. Docklight uses different colors and font

types to make it easy to distinguish between data transmitted by device 1 or device 2. The colors and font types can be chosen in the Display tab of the **Tools > Options...** dialog.

TIP: The Snapshot Function allows you to locate a rare sequence or error condition in a communication protocol with a large amount of data.

TIP: See the sections How to Increase the Processing Speed... and How to Obtain Best Timing Accuracy to learn how to adjust Docklight for applications with high amounts of data, or increased timing accuracy requirements.

5.4 Catching a Specific Sequence and Taking a Snapshot of the Communication

When monitoring serial communications between two devices, you might want to test for a rare error and the interesting parts would be just the serial communication before and after this event. You could look for this situation by logging the test and searching the log files for the characteristic error sequence. This could mean storing and analyzing several MB of data when you are actually just looking for a few bytes though, if they appeared at all. As an alternative, you can use the Snapshot feature as described below.

Preconditions

- Docklight is ready to run a test as described in the previous use cases, e.g. monitoring serial communications between two devices.

Taking a snapshot

A) Defining a trigger for the snapshot

1. Define the sequence that appears in your error situation as a Receive Sequence.
2. Check the **Trigger** tab in the "action" part of the Receive Sequence dialog: The trigger option must be enabled if this is the sequence that you want to track down.

NOTE: Do not forget to disable the trigger option for all other Receive Sequences that should be ignored in your test so that they do not trigger the snapshot.

B) Creating a snapshot

Click on the **Snapshot** button of the toolbar. Docklight will start communications, but will not display anything in the communication window. If the trigger sequence is detected, Docklight will display communication data before and after the trigger event. Further data is processed, until the trigger sequence is located roughly in the middle of the communication window. Docklight will then stop communication and position the cursor at the trigger sequence.

5.5 Logging and Analyzing a Test

Preconditions

- Docklight is ready to run a test as described in the previous use cases, e.g. Testing a serial device or a protocol implementation

Logging the test

Click on the **Start Logging** button on the main toolbar.

A dialog window will open for choosing log file settings.

For each representation (ASCII, HEX, ...), a separate log file may be created. Choose at least one representation. Log files will have a ".txt", ".htm" or ".rtf" file extension, depending on your file format choice. Docklight also adds the representation type to the file name to distinguish the different log files. E.g. if the user specifies "Test1" as the base log file name, the plain text ASCII file will be named "Test1_asc.txt", whereas an RTF HEX log file will be named "Test1_hex.rtf".

Confirm your log file settings and start logging by clicking the **OK** button.

To stop logging and close the log file(s), click the **Stop Logging** button on the main toolbar. Unless the log file(s) have been closed, it is not possible to view their entire contents.

5.6 Checking for Sequences With Random Characters (Receive Sequence Wildcards)

Many serial devices support a set of commands to transmit measurement data and other related information. In common text-based protocols the response from the serial device consists of a fixed part (e.g. "temperature="), and a variable part, which is the actual value (e.g "65F"). To detect all these responses correctly in the serial data stream, you can define Receive Sequences containing wildcards.

Take, for example, the following situation: A serial device measures the temperature and periodically sends the actual reading. Docklight shows the following output:

```
07/30/2012 10:20:08.022 [RX] - temperature=82F<CR>
07/30/2012 10:22:10.558 [RX] - temperature=85F<CR>
07/30/2012 10:24:12.087 [RX] - temperature=93F<CR>
07/30/2012 10:26:14.891 [RX] - temperature=102F<CR>
...
```

Defining an individual Receive Sequence for every temperature value possible would not be a practical option. Instead you would define one Receive Sequence using wildcards. For example:
t | e | m | p | e | r | a | t | u | r | e | = | ? | # | # | F | r
("r" is the terminating <CR> Carriage Return character)

This ReceiveSequence would trigger on any of the temperature strings listed above. It allows a 1-3 digit value for the temperature (i.e. from "0" to "999"). The following step-by-step example describes how to define the above sequence. See also the additional remarks at the end of this section for some extra information on '#' wildcards.

NOTE: See Calculating and Validating Checksums on how to receive and validate checksum data, e.g. CRCs. There are no wildcards required for checksum areas. Instead, use some default character values, e.g. "00 00" in HEX representation.

Preconditions

- Docklight is ready to run a test as described in the previous use cases, e.g. testing a serial device or a protocol implementation.
- The serial device (the temperature device in our example) is operating.

Using Receive Sequences with wildcards

A) Preparing the project
Create a new Docklight project and set up all communication parameters.

B) Defining the Receive Sequences used
1. Create a new Receive Sequence. Enter a **Name** for the sequence.
2. Enter the fixed part of your expected answer in the **Sequence** section. For our example you would enter the following sequence in ASCII mode:
   t | e | m | p | e | r | a | t | u | r | e | =
3. Open the popup / context menu using the **right mouse button**, and choose **Wildcard '?' (matches one character)** to insert the first wildcard at the cursor position. Add two '#' wildcards using the popup menu **Wildcard '#' (matches zero or one character)**. The sequence now looks like this:
   t | e | m | p | e | r | a | t | u | r | e | = | ? | # | #
4. Enter the fixed tail of our temperature string, which is a letter 'F' and the terminating <CR> character. You can use the default control character shortcut **Ctrl+Enter** to enter the <CR> / ASCII code 13. The sequence is now:
   t | e | m | p | e | r | a | t | u | r | e | = | ? | # | # | F | r
5. Specify an **Action** to perform after a temperature reading has been detected.
6. Click **OK** to add the new sequence to the Receive Sequence list.

NOTE: To distinguish the wildcards '?' and '#' from the regular question mark or number sign characters (decimal code 63 / 35), the wildcards are shown on a different background color within the sequence editor.

C) Running the test
Start Docklight by choosing **Run >** **Start Communication**.

Docklight will now detect any temperature reading and perform the specified action.

Additional notes on '#' wildcards

1. '#' wildcards at the end of a Receive Sequence have no effect. The Receive Sequence "HelloWorld###" will behave like a Receive Sequence "HelloWorld".

2. A "match inside a match" is not returned: If a Receive Sequence "Hello########World" is defined, and the incoming data is "Hello1Hello2World", the Receive Sequence detected is "Hello1Hello2World", not "Hello2World".

Receive Sequence comment macros

Macro keywords can be used in the Edit Receive Sequence **> 3 - Action > Comment** text box, to create Docklight comment texts with dynamic data, e.g. the actual data received.

Macro	Is Replaced By
%_S	BELL signal. Produce a 'beep sound', depending on your *Windows* sound scheme.
%_L	Line break
%_T	Time stamp for the data received

%_C	Docklight channel no. / data direction (1 or 2) for the data received
%_X	The channel name or channel alias that corresponds to the data direction %_C. E.g. "RX", "TX" or "COM5".
%_I	Receive Sequence List Index, see the Dialog: Edit Receive Sequence
%_N	Receive Sequence Name
%_A	The actual data that triggered this Receive Sequence. Use ASCII representation
%_H	Same as %_A, but in HEX representation
%_D	Same as %_A, but in Decimal representation
%_B	Same as %_A, but in Binary representation
%_A(1,4)	Extended syntax: Insert only the first 4 characters of this Receive Sequence (start with Character No. 1, sequence length = 4).
%_H(3,-1)	Extended Syntax: Insert everything from the third character until the end of the sequence (length = -1). Use HEX representation.

Example:
For a Receive Sequence as described above (t | e | m | p | e | r | a | t | u | r | e | = | ? | # | # | F | r), you could define the following comment text:

New Temp = %_L %_A(13, -3) °F

Docklight output could then look like this:

```
10/30/2012 10:20:08.022 [RX] - temperature=82F<CR>
 New Temp =
 82 °F
10/30/2012 10:22:10.558 [RX] - temperature=85F<CR>
 New Temp =
 85 °F
10/30/2012 10:24:12.087 [RX] - temperature=93F<CR>
 New Temp =
 93 °F
```

5.7 Saving and Loading Your Project Data

The project data includes:

- Send Sequences
- Receive Sequences
- Additional Project Settings: communication mode, COM ports used, COM port settings (baud rate, parity, ...)
- Documentation contents

The project is saved in a Docklight project file (**.ptp** file) using the menu **File >** **Save Project** or **File > Save Project As...**

It is generally recommended to save your project before starting a test run.

NOTE: Saving your project only stores the project's sequences, settings, and Documentation Area data. If you want to save a log of the communication during a test run, see section logging and analyzing a test.

Loading a project is done using the **File >** **Open Project...** menu.

Working with Docklight (Advanced)

6 Working with Docklight (Advanced)

6.1 Sending Commands With Parameters (Send Sequence Wildcards)

When testing a serial device, the device will most likely support a number of commands that include a parameter.

Example: A digital camera supports a command to set the exposure time. For setting the exposure time to 25 milliseconds, you need to send the following sequence:
e | x | p | | 0 | 2 | 5 | r ("r" is a terminating <CR> Carriage Return character)

To avoid defining a new Send Sequence for every exposure time you want to try, you can use a Send Sequence with wildcards instead:
e | x | p | | ? | ? | ? | r

The following step-by-step example describes how to define an exposure time command with a parameter and use a different exposure value each time the sequence is sent.

Preconditions

- Docklight is ready to run a test as described in testing a serial device or a protocol implementation.

Performing the test using commands with parameters

A) Preparing the project
Create a new Docklight project and set up all communication parameters.

B) Defining the commands used
1. Create a new Send Sequence. Enter a **Name** for the sequence.
2. Enter the fixed part of your command in the **Sequence** section. For our example you would enter the following sequence in ASCII mode:
   e | x | p | |
3. Now open the context menu using the **right mouse button**, and choose **Wildcard '?' (matches one character) F7** to insert one wildcard at the cursor position. In our example we would have to repeat this until there are three '?' wildcards for our three-digit exposure time. The sequence now looks like this:
   e | x | p | | ? | ? | ?
4. Now add the terminating <CR> character, using the default control character shortcut **Ctrl+Enter.** The example sequence now is
   e | x | p | | ? | ? | ? | r
5. Click **OK** to add the new sequence to the Send Sequence list.

Repeat steps 1 - 5 to define other commands needed to perform your test.

NOTE: To distinguish a '?' wildcard from a question mark ASCII character (decimal code 63), the wildcard is shown on a different background color within the sequence editor.

C) Sending a command to the serial device
1. Use the  **Send** button to open the serial communication port and send one command to the serial device.

2. The communication pauses and the Send Sequence Parameter dialog pops up, allowing you to enter the parameter value. In our example, an exposure time, e.g. "025".
3. Confirm by pressing **Enter**. The sequence is now sent to the serial device.

It is possible to define commands with several parameters, using several wildcard areas within one sequence. The Send Sequence Parameter dialog will then appear several times before sending out a sequence.

NOTE: If you are using **Wildcard '?'**, you must provide exactly one character for each '?' when sending the sequence. For variable-length parameters use **Wildcard '#' (matches zero or one character) F8**.

NOTE: You cannot use a Send Sequence with wildcards as an automatic answer for a Receive Sequence (see Action).

NOTE: If your Send Sequence requires a checksum, you can define it as described in Calculating and Validating Checksums. The checksum is calculated after the wildcard/parameter area has been filled with the actual data, then the resulting sequence data is handed over to the send queue.

6.2 How to Increase the Processing Speed and Avoid "Input Buffer Overflow" Messages

When monitoring serial communications between two devices, Docklight cannot control the amount of incoming data. Since Docklight applies a number of formatting and conversion rules on the serial data, only a limited number of bytes per seconds can be processed. There are numerous factors that determine the processing speed, e.g. the PC and COM devices used, the Display Settings, and the Receive Sequence Actions defined. It is therefore not possible to specify any typical data rates.

The most time-consuming task for Docklight is the colors&font formatting applied by default (see the Docklight Display Options). If Docklight cannot keep up with formatting the incoming data, it will automatically switch to the simpler Plain Text Mode.

If this is still not fast enough to handle the incoming data, Docklight will add the following message in the Communication Window output and log files.

```
DOCKLIGHT reports: Input buffer overflow on COM1
```

TIP: Search for this message using the **Find Sequence in Communication Window...** (Ctrl + F) function.

If you are experiencing the above behavior, Docklight offers you several ways to increase the data throughput.

1. Simplify the display output:
   - Deactivate all unneeded Display Modes in the **Options...** dialog
   - Use Plain Text Mode right from the start (see the automatic switch behavior described above).
   - If you are using ASCII mode, disable the Control Characters Description option

2. Log the communication data to a plain text file instead of using the communication window(s):
   - Use the "plain text" Log File Format
   - Create only a log file for the Representation (ASCII / HEX / Decimal / Binary) you actually need

- Disable the communication windows while logging, using the High Speed Logging option

3. Use the Communication Filter from the **Project Settings...** dialog, and disable the original serial data for one or both communication directions. This is especially useful if you actually know what you are looking for and can define one or several Receive Sequences for this pieces of data. These Receive Sequences can print a comment each time the sequence appears in the data stream so you still know what has happened, even if the original serial data is not displayed by Docklight.

6.3 How to Obtain Best Timing Accuracy

Many RS232 monitoring applications – including Docklight – can only provide limited accuracy when it comes to time tagging the serial data. As a result, data from the two different communication directions can be displayed in chronologically incorrect order, or several telegrams from one communication direction can appear as one chunk of data.

This behavior is not caused by poor programming, but is rather characteristic for a PC/Windows system, and the various hardware and software layers involved. Unspecified delays and timing inaccuracies can be introduced by:

- The COM device's chipset, e.g. the internal FIFO (First-In-First-Out) data buffer.
- The USB bus transfer (for USB to Serial converters).
- The serial device driver for Windows.
- The task/process scheduling in a multitasking operating system like Windows.
- The accuracy of the date/time provider.

Docklight comes with a very accurate date/time provider with milliseconds granularity, but it still needs to accept the restrictions from the hardware and software environment around it.

Here is what you can do to minimize additional delays and inaccuracies and achieve a typical time tagging accuracy of 5 milliseconds or better:

1. Get our Docklight Tap for lowest USB-related latency times. Or use on-board RS232 ports, if still available on your PC.

2. Choose External / High Priority Process Mode in the **Tools > Expert Options...** dialog.

3. When monitoring high amounts of data, use the recommendations from the previous section How to Increase the Processing Speed... to avoid input buffer overflows and that the computer become irresponsive because of high CPU usage.

NOTE: The **Expert Options...** recommended above will change the overall system balance and must be used with care. Best results can be achieved only when Docklight is **Run as administrator**. Please make sure you understood the remarks and warning in the documentation.

4. As an alternative to the above 1.-3.: Use our Docklight Tap Pro or Docklight Tap 485 accessories which use their own Embedded time provider and eliminate PC-based inaccuracies altogether.

6.4 Calculating and Validating Checksums

Many communication protocols include additional checksum fields to ensure data integrity and detect transmission errors. A common algorithm is the CRC (Cyclic

Redundancy Code), which is used in different variations for different protocols. The following step-by-step example describes how to set up on-the-fly checksum calculation for a Send Sequence, and how to enable automatic validation of a checksum area within a Receive Sequence.

TIP: For a working example to address a Modbus slave device, see the tutorial Modbus RTU With CRC checksum.

Preconditions

You know the checksum specification for the protocol messages:

- Which area of the sequence data is guarded by a checksum?
- Where is the checksum located? (Usually at the end of the sequence.)
- What checksum algorithm should be used? (Most likely one of the standard CRC types, or a simple MOD256 sum.)

Using Send Sequences with automatic checksum calculation

A) Defining a Send Sequence that includes a checksum

1. Create a new Send Sequence. Enter a **Name** for the sequence.

2. Enter the Sequence part of your message in the **Sequence** section. For example, here we use a very simple HEX message as our sequence:
   01 | 02 | 03 | 04 | **??**

   Use the context menu via **right mouse button** or **F7** to create the **??** wildcard.

   NOTE: See also the Send Sequence Parameter section for more information on wildcards and parameters.

3. Now add one additional 00 value as a placeholder for the checksum.
   01 | 02 | 03 | 04 | ?? | **00**

   NOTE: In a Send Sequence, you can use any character code from 00-FF as a placeholder at the positions where the calculated checksum should be inserted later. This is different from the way it works in a Receive Sequence, where you use ?? wildcards. See the Receive Sequence example below.

4. Go to the **Additional Settings** | **Checksum** tab and define the checksum. For example, here we chose to use **MOD256** from the dropdown list.

   NOTE: The text field for Checksum allows comments. Everything behind a **#** character is just a comment. You can add your own comments to describe what this checksum is about.

5. Click **OK** to add the new sequence to the Send Sequence list.

B) Performing the test

6. Use the  **Send** button to send one of the predefined commands. Enter a parameter value, e.g. 05.

Before sending the data, Docklight calculates the actual MOD256 checksum. The result goes to the specified checksum position. For MOD256 this is the last character position by default, which means that the 00 placeholder is overwritten with the checksum result.

If we use 05 as a parameter when sending the sequence, the data sent by Docklight will look like this:

```
18.06.2015 11:07:23.251 [TX] - 01 02 03 04 05 0F
```

The placeholder has been replaced by the sum over the message bytes:
1 + 2 + 3 + 4 + 5 = 15 or Hex 0F.

Using Receive Sequences with automatic checksum validation

A) Defining a Receive Sequence with checksum

1. Create a new Receive Sequence. Enter a **Name** for the sequence.
2. Enter the **Sequence** data, including a wildcard area for both a random payload byte, plus a wildcard for the checksum. We use the same telegram as in the above Send Sequence example:
   01 | 02 | 03 | 04 | ?? | **00**.
3. Go to the **Action | Comment** tab and enter the following text: **Correct checksum**
4. Go to the **Checksum** tab and pick **MOD256** in the left dropdown list.
5. Keep the **Detect Checksum OK** option. It means that the Receive Sequence is only triggered if the MOD256 checksum byte in the received data is correct.
5. Click **OK** to confirm the changes

B) Running the test

6. Start communications and send some data telegrams to your Docklight application / COM port.

The Communication Window output could look like this:

```
15.02.2016 17:43:28.072 [RX] - 01 02 03 04 05 0F  Correct
checksum

15.02.2016 17:43:31.870 [RX] - 01 02 03 04 0F 19  Correct
checksum

15.02.2016 17:43:35.833 [RX] - 01 02 03 04 10 1A  Correct
checksum
```

NOTE: This example showed how to define a Receive Sequence that is triggered by data telegrams with correct checksum only. It is also possible to do the opposite: detecting a checksum error. Go to the **Checksum** tab and change the option **Detect Checksum OK** to **Checksum Wrong**.

6.5 Controlling and Monitoring RS232 Handshake Signals

The Docklight project settings for Flow Control support offer a Manual Mode that allows you to set or reset the RTS and DTR signals manually by clicking on the corresponding indicator. The following section describes how to use the **Function Character '!' (F11 key)** to change the RTS and DTR signals temporarily within a Send Sequence, or detect changes for the CTS, DSR, DCD or RI lines using a Receive Sequence.

Preconditions

- Docklight is ready to run a test as described in testing a serial device or a protocol implementation.
- Flow Control Support is set to "Manual" in the project settings.
- The Docklight project already contains one or several Send Sequences, but there is an additional requirement for changing RTS / DTR signals while sending.

Implementing RTS/DTR signal changes

For our example we assume that we are using a RS485 converter which requires RS485 Transceiver Control, but uses the DTR signal instead of RTS for switching between "transmit" and "receive" mode. We further assume there is already a "Test" Send Sequence which looks like this in ASCII mode:
T | e | s | t

A) Modifying the existing Send Sequence

1. Open the Edit Send Sequence dialog.
2. Switch the **Edit Mode** to **Decimal**. Our "Test" example looks like this in decimal mode:
   084 | 101 | 115 | 116
3. Insert an RTS/DTR function character at the beginning: Press **F11**, or open the context menu using the **right mouse button** and choose **Function character '!' (RTS and DTR signals)** . The example sequence now reads:
   ! | 084 | 101 | 115 | 116
4. Add the new RTS/DTR state as a decimal parameter value (see below). In our example we need the DTR signal set to high. We choose "002" as the parameter value, so the sequence is now:
   ! | 002 | 084 | 101 | 115 | 116
5. Add a RTS/DTR function character at the end of the sequence, and use "000" as parameter value to reset the DTR signal low. The sequence data is now:
   ! | 002 | 084 | 101 | 115 | 116 | ! | 000
6. Click **OK** to confirm the changes

NOTE: To distinguish a '!' RTS/DTR function character from a exclamation mark ASCII character (decimal code 33), the RTS/DTR function character is shown on a different background color by the sequence editor.

NOTE: The character after a RTS/DTR function character is used to set the RTS / DTR signals and is not sent to the serial device (see parameter values below).

B) Sending the data with additional DTR control

1. Send the test sequence using the  **Send** button.

Docklight will now set the DTR signal to high, send the ASCII sequence "Test" and then reset DTR.

NOTE: The RTS/DTR indicators will indicate any changes of the RTS or DTR state. However, in the above example the DTR is set and reset very quickly, so the DTR indicator will probably not give any visual feedback. If you want to actually "see" the DTR behavior, try introducing a small inter-character delay.

Function character '!' (F11) - setting RTS and DTR

Character Value (Decimal Mode)	RTS	DTR
000	Low	Low
001	High	Low
002	Low	High
003	High	High

Temporary parity changes / 9 bit applications

Some protocols and applications require a 9th data bit, e.g. for device addressing on a bus. The only way to talk to such devices using a standard UART with maximum 8 data bits is to use serial settings that include a parity bit and change this parity bit temporarily within a Send Sequence. The function character '!' supports additional parameter values for this purpose:

Character Value (Decimal Mode)	Parity
016	No parity
032	Odd parity
048	Even parity
064	Mark. Set parity bit to logic '1'
080	Space. Set parity bit to logic '0'

The new parity settings are applied starting with the next regular character, both on the TX and the RX side. The parity is switched back to the original Communication Settings after the Send Sequence has been completely transmitted.

NOTE: The most useful parameters for this function character are the "Mark" and "Space" settings, because they allow you to set the parity bit to a defined value that effectively serves as a 9th data bit.

NOTE: It is recommended to set the Parity Error Character to "(ignore)", so you can evaluate incoming data in both cases, 9th bit = high and 9th bit = low.

TIP: See also the **SwitchParityDemo.ptp** sample project (folder **Extras\ParitySwitch_9BitProtocols** in your \Samples directory).

Function character '!' (F11) - detecting handshake signal changes (CTS, DSR, DCD or RI)

Docklight Scripting detects changes of the handshake signals CTS, DSR, DCD or RI, but in normal operation these changes are not visible in the Docklight Communication Window (similar to a Break State).

Using the function character '!' you can make these changes visible, and/or define an action after detecting such changes. The function character '!' supports the following parameter values for this purpose:

Character Value (Decimal Mode)	Handshake Signal
001	CTS = High
002	DSR = High
004	DCD = High
008	RI (Ring Indicator) = High

Example Receive Sequence definitions in Decimal Edit Mode:

Receive Sequence (Decimal Mode)	Description
! | 001	triggers when CTS=high, all other signals low
! | 006	triggers when CTS=low, DSR=high, DCD=high, RI=low
! | ???	triggers on any change of the status lines

For the following example we assume that Docklight is ready to run a test as described in testing a serial device or a protocol implementation and Flow Control Support is set to "Manual" in the project settings.

A) Create a new Receive Sequence for detecting handshake signal changes.

1. Open the Edit Receive Sequence dialog.
2. Switch the **Edit Mode** to **Decimal**.
3. Insert a 'signals' function character at the beginning: Press **F11**, or open the context menu using the **right mouse button** and choose **Function character '!' (CTS/DSR/DCD/RI changes)** .
4. Add the handshake state as a decimal parameter value (see above). In our example we want to detect when CTS is high, while all other signals are low. This means we need to enter "001" as the parameter value, so the sequence is now:
   ! | 001
5. Specifiy a **Comment** for this sequence, e.g. "[CTS = high, DSR/DCD/RI = low]"
6. Click **OK** to confirm the new sequence

B) Start the test and confirm that Docklight now detects when the CTS line changes from low to high.

Function character '^' (F12) - bitwise comparisons

The Function Character '^' can be added by pressing **F12** in the Edit Receive Sequence dialog. After the '^' character, two additional character values specify which bits to check (*mask*) and which values to expect for these bits (*value*).

Receive Sequence (HEX Mode)	Description
^ | *mask* | *value*	Is a match for the next character received, when the following is true: ((*nextCharacterReceived* XOR *value*) AND *mask*) = 0 In other words - the '^' character picks only the bits marked in *mask* and compares them with the corresponding bits in *value*. See below for examples.
^ | 0F | 05	Is a match, when for the next character the following is true: Bit 0 = 1 Bit 1 = 0 Bit 2 = 1 Bit 3 = 0 Bit 4-7 = (don't care)
! | ^ | 04 | 04	This Receive Sequence triggers when the new handshake signal state says DCD = High. All other handshake signals can have any state.

	NOTE: This Receive Sequence will trigger for any change of any handshake signal, in case DCD still remains High.

TIP: This extension is also demonstrated in the Docklight Scripting example project **Docklight_TapPro_Demo.ptp** (see the folder **Extras\TapPro** in your \Samples directory)

6.6 Creating and Detecting Inter-Character Delays

Some applications, especially microcontroller applications without a dedicated serial data buffer, require an extra delay between individual characters to avoid buffer overflows and allow the microcontroller to execute other code.

In Docklight you can implement inter-character delays by inserting one or several **Function Characters '&' (F9 key)** in your Send Sequence data, followed by a character specifying the desired delay time from 0.01 seconds to 2.55 seconds.

You can also use the '&' delay character inside a Receive Sequence to specify a minimum silent time where no further characters should be received. This is useful for detecting pauses in the data stream that indicate the beginning/end of a telegram, especially for protocols where there is no dedicated start or end character.

Preconditions

- Docklight is ready to run a test as described in testing a serial device or a protocol implementation.
- The Docklight project already contains one or several Send Sequences, but an additional delay at certain character positions is required.

Sending Data With Inter-Character Delays

As an example, we use a microcontroller application which understands a "get" command. In ASCII Mode, the Send Sequence would be:
g | e | t | r ("r" is a terminating <CR> Carriage Return character)

The following steps describe how to add an additional delay of 20 milliseconds between each character and avoid buffer overflows on the microcontroller side.

A) Modifying the existing Send Sequence

1. Open the Edit Send Sequence dialog.
2. Switch **Edit Mode** to **Decimal**. Our "get" example looks like this in decimal mode:
   103 | 101 | 116 | 013
3. Insert a delay function character between the first and the second character: Press **F9**, or open the context menu using the **right mouse button**, and choose **Function character '&' (delay...** . The example sequence now reads:
   103 | & | 101 | 116 | 013
4. Add the delay time: In this example a decimal value of 002 (20 milliseconds) after the "&" function character is added. The sequence is now:
   103 | & | 002 | 101 | 116 | 013
5. Insert a delay between all other inter-character positions: the delay character and delay time can be copied using **Ctrl+C**, and pasted in the desired positions using **Ctrl+V**. Our example sequence finally reads:
   103 | & | 002 | 101 | & | 002 | 116 | & | 002 | 013
   Or back in ASCII Mode:
   g | & | □ | e | & | □ | t | & | □ | r
6. Click **OK** to confirm the changes

NOTE: To distinguish a '&' delay character from a regular ampersand ASCII character (decimal code 38), the delay function character is shown on a different background color by the sequence editor.

NOTE: The character after a delay function character is interpreted as the delay time and is not sent to the serial device.

B) Sending the command to the microcontroller application

1. Send the modified Send Sequences using the  **Send** button.

Docklight will send out the same data as before, but leave additional timing gaps as specified by the delay characters. The communication display will show the same communication data as without the delays.

NOTE: Docklight's accuracy for delay timing is limited because it has no control over the UART's internal TX FIFO buffer. The specified delay times for the '&' delay function character are minimum values. Measured delay values are significantly higher, because Docklight always waits a minimum time to ensure the UART TX FIFO buffer is empty. Also, the display format and the additional performance settings affects the timing. If you have more specific requirements on Send Sequence timing and need to control the Docklight "wait time" as well as your UART FIFO settings, please contact our e-mail support.

Pause detection using a Receive Sequence

Docklight already offers the Pause detection... display option to insert additional time stamps or line breaks after communication pauses.

If you require not only visual formatting, but need to define actions after a minimum pause, or simply make sure the Receive Sequence detection algorithm starts anew after a pause, you can add the delay function character to your Receive Sequence definition.

In most applications the best place for the delay function character will be at the beginning of the Receive Sequence, before the actual receive data characters. You can also create a Receive Sequence that contains a delay/pause definition only, and no actual serial data. This can be very useful for implementing timing constraints, e.g. resetting the telegram detection after a pause occurred.

6.7 Setting and Detecting a "Break" State

Some serial application protocols (e.g. LIN) make use of the so-called Break state for synchronization purposes. Docklight Scripting supports sending a "break" within a Send Sequence and detecting a "break" state using a Receive Sequence definition. "break" signals are added to your sequence definition by inserting a **Function Character '%' (F10 key)**. A Docklight "break" signal has a minimum length of 15 * <nominal bit length>.

Preconditions

- Docklight is ready to run a test as described in testing a serial device or a protocol implementation.

- The Docklight project already contains one or several Send Sequences, but signalling or detecting a "break" state is also required.

Sending a "Break" state

We assume there is already a "Test" Send Sequence which looks like this in ASCII mode:
T | e | s | t

1. Open the Edit Send Sequence dialog.
2. Insert a "Break" function character at the beginning: Press **F10**, or open the context menu using the **right mouse button**, and choose **Function character '%' (break signal)** . The example sequence now reads:
   % | T | e | s | t
3. Click **OK** to confirm the changes
4. Send the test sequence using the  **Send** button.

The TX line will go to Space (logical 0) for at least 15 bit durations, then the "Test" ASCII sequence will be transmitted. The "break" character does not appear in the communication window display.

Detecting a "Break" state

Received "break" signals are not displayed in the communication window, because they are not part of the actual data sequence. Nonetheless, it is possible to define a Receive Sequence including a "break" function character.

1. Create a new Receive Sequence. Enter a **Name** for the sequence.
2. Add a **Function character '%' (break signal)** for the **Sequence** data.
3. Enter a Receive Sequence **Action**, for example printing the comment "BREAK detected"
4. Click **OK** to confirm the changes
5. Start communications.

Docklight will now add `BREAK detected` to the communication window display each time a break signal is detected.

NOTE: After detecting a break signal, an additional <NUL> character (decimal code 0) may appear in the received data stream. This behavior cannot be controlled by Docklight, it depends on how the serial UART of your PC's COM port interpretes the break state.

NOTE: If you need to implement a Receive Sequence that checks for a break signal followed by additional data, keep in mind that Docklight cannot tell the exact position of the break signal within the data stream. The break signal will sometimes show up earlier in the data stream, but never later than the actual position. To define a Receive Sequence that safely triggers on break + specific data, you can use the following workaround: Insert some '#' (zero or one character) wildcards between the break character and the additional data. The resulting Receive Sequence could look like this:
% | # | # | # | # | # | # | # | T | e | s | t

Examples and Tutorials

7 Examples and Tutorials

This chapter describes two sample projects that demonstrate some of Docklight's basic functions. The corresponding Docklight project files (**.ptp** files) can be found in the **\Samples** folder within the Docklight installation directory (e.g. **C:\Program Files\FuH\Docklight V2.3\Samples**).

TIP: The **\Samples** folder can also be reached via the Docklight Welcome screen (menu **Help > Welcome Screen and Examples...**).

NOTE: If you are logged on with a restricted user account, you will not have permission to make any changes in the program files directory. In this case, saving a project file or any other data into the **\Samples** folder will produce an error.

NOTE: For additional sample projects and Application Notes, see also our online resources at https://docklight.de/examples/.

7.1 Testing a Modem - Sample Project: ModemDiagnostics.ptp

The Docklight project **ModemDiagnostics.ptp** can be used to perform a modem check. A set of modem diagnostic commands are defined in the Send Sequences list.

This is a simple example for Testing a serial device or a protocol implementation. The sample project uses the communication settings listed below. This should work for most standard modems.

Communication Mode	Send/Receive
COM Port Settings	9600 Baud, No parity, 8 Data Bits, 1 Stop Bit

Getting started

- Use the *Windows* **Device Manager** to find out which **COM Port** is a modem device. This demo project may be used with any AT-compatible modem available on your PC, e.g. a built-in notebook modem, or a GSM or Bluetooth modem driver than can be accessed through a virtual COM port.

TIP: For a simple test without specialized hardware, add your mobile phone as **Bluetooth Device** on your *Windows* PC. Then find your phone in the *Windows* **Devices and Printers** list. Right-click on it, choose **Properties** and go to the **Hardware** tab. In the **Device Functions** list it should mention the modem related COM Ports.

- Go to the **Project Settings...** dialog and make sure you have selected the same COM Port for **Send/Receive on comm. channel**.
- Press the **Start Communication** button in the toolbar.
- Try sending any of the predefined modem commands by pressing the **Send** button

You should now receive a response from your modem, e.g. "OK" if your command was accepted, a model identification number, etc. The response will vary with the modem model.

After sending several sequences, the Docklight communication window could look like this:

```
07.02.2013 18:17:54.083 [TX] - ATQ0V1E0<CR><LF>
```

```
07.02.2013 18:17:54.107 [RX] - ATQ0V1E0<CR><LF>
<CR><LF>
OK<CR><LF>

07.02.2013 18:18:00.511 [TX] - ATI2<CR><LF>

07.02.2013 18:18:00.747 [RX] - <CR><LF>
V 11.10<CR><LF>
13-05-11<CR><LF>
RM-721<CR><LF>
(c) Nokia            <CR><LF>
<CR><LF>
OK<CR><LF>

07.02.2013 18:18:01.393 [TX] - ATI3<CR><LF>

07.02.2013 18:18:01.421 [RX] - <CR><LF>
Nokia C2-01<CR><LF>
<CR><LF>
OK<CR><LF>
...
```

Further Information

The Send Sequences list includes the following standard AT modem commands:

Send Sequence	Description / Modem Response
ATQ0V1E0	Initializes the query.
AT+GMM	Model identification (ITU V.250 recommendation is not supported by all modems).
AT+FCLASS=?	Fax classes supported by the modem, if any.
AT#CLS=?	Shows whether the modem supports the Rockwell voice command set.
ATI<n>	Displays manufacturer's information for <n> = 1 through 7. This provides information such as the port speed, the result of a checksum test, and the model information. Check the manufacturer's documentation for the expected results.

The \Samples folder also contains a log file **ModemDiagnostics_Logfile_asc.txt**. It shows a test run where the above Send Sequences were sent to a real modem.

7.2 Reacting to a Receive Sequence - Sample Project: PingPong.ptp

The Docklight project **PingPong.ptp** is a simple example for how to define and use Receive Sequences.

Getting started

- Go to the **Project Settings...** dialog and choose a COM port.
- Apply a simple loopback to this COM port: Connect Pin 2 (RX) with Pin 3 (TX). See RS232 SUB D9 Pinout.
- Now press the **Send** button for either of the two Send Sequences. Communication is started and the Send Sequence is transmitted. It will of course be instantly received on the COM port's RX line.

Docklight will detect the incoming data as being one of the defined Receive Sequences. It will then perform the action predefined for this event, which is sending out another sequence. As a result, Docklight will send out alternating Send Sequences - "Ping" and "Pong".

- Use the **Stop communication** button to end the demo.

The Docklight communication display should look similar to this:

```
3/8/2009 16:25:44.201 [TX] - ----o Ping
3/8/2009 16:25:44.216 [RX] - ----o Ping "Ping" received
3/8/2009 16:25:44.218 [TX] - o---- Pong
3/8/2009 16:25:44.233 [RX] - o---- Pong "Pong" received
3/8/2009 16:25:44.236 [TX] - ----o Ping
3/8/2009 16:25:44.251 [RX] - ----o Ping "Ping" received
3/8/2009 16:25:44.254 [TX] - o---- Pong
3/8/2009 16:25:44.268 [RX] - o---- Pong "Pong" received
3/8/2009 16:25:44.271 [TX] - ----o Ping
3/8/2009 16:25:44.286 [RX] - ----o Ping "Ping" received
3/8/2009 16:25:44.289 [TX] - o---- Pong
3/8/2009 16:25:44.303 [RX] - o---- Pong "Pong" received
3/8/2009 16:25:44.307 [TX] - ----o Ping
3/8/2009 16:25:44.322 [RX] - ----o Ping "Ping" received
3/8/2009 16:25:44.324 [TX] - o---- Pong
...
```

See also the corresponding log files in the \Samples folder (**PingPong_Logfile_asc.htm** and **PingPong_Logfile_hex.htm**).

Further Information

This demo project can also be run in three alternative configurations:

1. Run two Docklight applications on the same PC using different COM ports. The two COM ports are connected using a simple null modem cable.
2. Instead of two RS232 COM ports and a null modem cable you can use a virtual null modem.
3. Use two PCs and run Docklight on each PC. Connect the two PCs using a simple null modem cable.

7.3 Modbus RTU With CRC checksum - Sample Project: ModbusRtuCrc.ptp

The Docklight project file **ModbusRtuCrc.ptp** demonstrates how to automatically calculate the CRC value required to send a valid Modbus RTU frame.

The project file uses the communication settings listed below, according to the Modbus implementation class "Basic".

Communication Mode	Send/Receive
Send/Receive on comm. channel	COM1
COM Port Settings	19200 Baud, Even parity, 8 Data Bits, 1 Stop Bit

Getting started

- Open the project file **ModbusRtuCrc.ptp** (menu Open Project ...). The file is located in the \Samples folder.
- Connect the PC's COM port to your Modbus network. Open the **Project Settings...** dialog and make sure you have selected the correct COM Port for **Send/Receive on comm. channel**.
- Click the ----> **Send** button in the **Read Input Register Slave=?..** line of the Send Sequence list
- Enter a slave number in the **Send Sequence Parameter** dialog, e.g. "01" for addressing slave no. 1.

After sending "Read Input Register" commands to slaves 1 - 4, the communication window could look like this:

```
23.09.2019 07:04:56.170 [TX] - 01 04 00 03 00 01 C1 CA
23.09.2019 07:04:56.282 [RX] - 01 04 02 FF FF B8 80
Detected Modbus Frame = 01 04 02 FF FF B8 80
SlaveID=01
FunctionCode=04
Addr/Data=02 FF FF
CRC=B8 80

 Input Register Answer: Slave=001 ValueHex=FFFF

23.09.2019 07:05:21.761 [TX] - 02 04 00 03 00 01 C1 F9
23.09.2019 07:05:21.873 [RX] - 02 04 02 7F 58 DC FA
Detected Modbus Frame = 02 04 02 7F 58 DC FA
SlaveID=02
FunctionCode=04
Addr/Data=02 7F 58
CRC=DC FA

 Input Register Answer: Slave=002 ValueHex=7F58

23.09.2019 07:05:35.713 [TX] - 03 04 00 03 00 01 C0 28
23.09.2019 07:05:35.824 [RX] - 03 04 02 01 0A 41 67
Detected Modbus Frame = 03 04 02 01 0A 41 67
SlaveID=03
FunctionCode=04
Addr/Data=02 01 0A
CRC=41 67

 Input Register Answer: Slave=003 ValueHex=010A

23.09.2019 07:05:51.677 [TX] - 04 04 00 03 00 01 C1 9F
23.09.2019 07:05:51.789 [RX] - 04 04 02 40 00 44 F0
Detected Modbus Frame = 04 04 02 40 00 44 F0
SlaveID=04
FunctionCode=04
Addr/Data=02 40 00
CRC=44 F0

 Input Register Answer: Slave=004 ValueHex=4000
```

The [RX] channel shows the responses from the Modbus slaves:
slave 1 responded value "-1",

slave 2 responded "32600",
slave 3 responded "266" and
slave 4 responded "16384".

NOTE: If you are using the Docklight Modbus example on a RS485 bus and do not see a device answer, check if your RS485 hardware interface automatically switches between transmit and receive mode, or you need to use the RS485 Transceiver Control option.

Further Information

- The CRC calculation is made according to the specifications for Modbus serial line transmission (RTU mode). Docklight's checksum function supports a "CRC-MODBUS" model for this purpose. See Calculating and Validating Checksums for more general information on implementing checksum calculations.
- If you do not have any Modbus slave devices available, you can use a software simulator. See the www.plcsimulator.org/ as originally mentioned on www.modbus.org, "Modbus Technical Resources", "Modbus Serial RTU Simulator". This simulator was used to produce the sample data shown above.

Reference

8 Reference

8.1 Menu and Toolbar

NOTE: Hotkeys are available for common menu and toolbar functions.

File Menu

New Project
Close the current Docklight project and create a new one.

Open Project ...
Close the current Docklight project and open another project.

Import Sequence List ...
Import all Send Sequences and Receive Sequences from a second Docklight project.

Save Project / Save Project As ...
Save the current Docklight project.

Print Project ...
Print the project data, i.e. the list of defined Send Sequences and Receive Sequences. The sequences are printed in the same representation (ASCII, HEX, Decimal or Binary) that is used in the Docklight main window. The representation may be chosen using the Options dialog window.

Print Communication ...
Print the contents of the communication window. The communication data is printed in the same representation that is currently visible in the communication window.

Exit
Quit Docklight.

Edit Menu

Edit Send Sequence List ...
Edit the Send Sequences list, i.e. add new sequences or delete existing ones.

Edit Receive Sequence List ...
Edit the Receive Sequences list, i.e. add new sequences or delete existing ones.

Swap Send and Receive Sequence Lists
Convert all Send Sequences into Receive Sequences and vice versa.

Find Sequence in Communication Window...
Find a specific sequence within the data displayed in the communication window. See the Find Sequence function.

Clear Communication Window
Delete the contents of the communications window. This applies to all four representations (ASCII, HEX, Decimal, Binary) of the communication window.

Run Menu

Start communication
Open the communication ports and enable serial data transfer.

Stop communication
Stop serial data transfer and close the communication ports.

Tools Menu

Start Communication Logging ...
Create new log file(s) and start logging the incoming/outgoing serial data. See logging and analyzing a test.

Stop Communication Logging
Stop logging and close the currently open log file(s).

Start Snapshot Mode
Wait for a trigger sequence and take a snapshot. See Catching a specific sequence...

Stop Snapshot Mode
Abort a snapshot and reenable the communication window display.

Keyboard Console On
Enable the keyboard console to send keyboard input directly.

Keyboard Console Off
Disable the keyboard console.

Minimize/Restore Documentation Area
Minimize the Documentation Area, or bring it back to regular size.

Minimize/Restore Sequence Lists
Minimize the Send/Receive Sequence lists, or bring them back to regular size.

Project Settings...
Select the current project settings (communication settings, flow control settings, communication filter...).

Options...
Select general settings (e.g. display).

Expert Options...
Select expert program options intended for advanced users and specific applications (e.g. high monitoring accuracy).

8.2 Dialog: Edit Send Sequence

This dialog is used to define new Send Sequences and edit existing ones (See also Editing and Managing Sequences).

Index
The index of the sequence displayed below. The first sequence has index 0 (zero).

1 - Name
Unique name for this sequence (e.g. "Set modem speaker volume"). This name is for referencing the sequence. It is not the data that will be sent out through the serial port. See "2 - Sequence" below.

2 - Sequence
The character sequence that will be transmitted through the serial port.

3 - Additional Settings

- **Repeat** - Check the "Send periodically..." option to define a sequence that is sent periodically. A time interval between 0.01 seconds and 9999 seconds can be specified.

NOTE: The *Windows* reference time used for this purpose has only limited precision. Time intervals < 0.03 seconds will usually not be accurate.

- **Checksum** - Perform automatic calculation of any type of checksum, including any type of CRC standard such as Modbus, CCITT, CRC32.

TIP: See Calculating and Validating Checksums for a general overview, and Checksum Specification for the text format used to define a checksum.

Wildcards
Wildcards can be used to introduce parameters into a Send Sequence that you wish to insert manually each time the sequence is sent. See section Sending commands with parameters for details and examples.

Control Character Shortcuts
Using keyboard shortcuts is a great help when editing a sequence that contains both printing characters (letters A-z, digits 0-9, ...) and non-printing control characters (ASCII code 0 to 31). Predefined shortcuts are:
Ctrl+Enter for carriage return / <CR> / decimal code 13
Ctrl+Shift+Enter for line feed / <LF> / decimal code 10

Use Options... --> Control Character Shortcuts to define other shortcuts you find useful.

Sequence Documentation
Add some documentation about this sequence here. This documentation is also shown in the main window when selecting the sequence in the Send Sequences list.

8.3 Dialog: Edit Receive Sequence

This dialog is used to define new Receive Sequences and edit existing ones (See also Editing and Managing Sequences).

Index
The index of the sequence displayed below. The first sequence has index 0 (zero).

1 - Name
Unique name for this sequence (e.g. "Ping received"). This name is for referencing the sequence. It is not the sequence received through the serial port. See "2 - Sequence" below.

2 - Sequence
The character sequence which should be detected by Docklight within the incoming serial data.

TIP: Special Function Characters are available for detecting inter-character delays, evaluating handshake signal changes or detecting a break state.

3 - Action
The action(s) performed when Docklight detects the sequence defined above.

You may choose from the following actions:

- **Answer** - After receiving the sequence, transmit one of the Send Sequences. Only Send Sequences that do not contain wildcards can be used as an automatic answer.
- **Comment** - After receiving the sequence, insert a user-defined comment into the communication window (and log file, if available). Various comment macros are available for creating dynamic comment texts.
- **Trigger** - Trigger a snapshot when the sequence is detected. This is an advanced feature described in the section Catching a specific sequence...
- **Stop** - Stop communications and end the test run.
- **Checksum** - Perform automatic validation of a checksum, including any type of CRC standard such as Modbus, CCITT, CRC32.
  Set the Checksum Specification, as well as what should be done with the result:
  Detect Checksum OK - the received data must have the same checksum than the calculated value from Docklight.
  Checksum Wrong - the opposite. A mismatching checksum constitutes a "sequence match".
  Both OK/Wrong - the sequence is always detected. The checksum area will contain all ASCII "1" (HEX 31) for a matching checksum, or ASCII "0" (HEX 30) for a wrong checksum.

TIP: See Calculating and Validating Checksums for a general overview, and Checksum Specification for the text format used to define a checksum.

Wildcards
Wildcards can be used to test for sequences that have a variable part with changing values (e.g. measurement or status values). See section Checking for sequences with random characters for details and examples.

Control Character Shortcuts
Using keyboard shortcuts is a great help when editing a sequence that contains both printing characters (letters A-z, digits 0-9, ...) and non-printing control characters (ASCII code 0 to 31). Predefined shortcuts are:
Ctrl+Enter for carriage return / <CR> / decimal code 13
Ctrl+Shift+Enter for line feed / <LF> / decimal code 10

Use Options... --> Control Character Shortcuts to define other shortcuts you find useful.

Sequence Documentation
Add some documentation about this sequence here. This documentation is also shown in the main window when selecting the sequence in the Send Sequences list.

8.4 Dialog: Start Logging / Create Log File(s)

Menu **Tools > Start Communication Logging ...**

Log file format
The available log formats are plain text (.txt), HTML for web browsers (.htm), or RTF Rich Text Format for *Microsoft* Word or Wordpad (.rtf).

- **Plain text file (.txt)** is a good choice if you expect your log files to become very large.

TIP: The *Windows* built-in Notepad editor can be very slow in opening and editing larger files. We recommend the popular Open Source editor Notepad++ as available at http://notepad-plus.sourceforge.net - it is a much faster and more powerful alternative.

NOTE: there is no size limit for Docklight log files besides the limits on your Windows PC. We have successfully tested Docklight in long-term monitoring / high volume

applications and created log files with several GB size without any stability issues.

- **HTML files (.htm)** are more comfortable to analyze, because they include all the visual formatting of the Docklight communication windows (colors, bold characters, italic characters). However, the disk size for such a file will be larger than for a plain text format, and large HTML files will slow down common web browsers.

  TIP: If you have specific requirements on the output format, you can customize the HTML output.

- **RTF Rich Text Format (.rtf)** is a good choice for both small and large log files with formatted text - both *Microsoft* Word and Wordpad can navigate through larger files fast and without appearing unresponsive.

NOTE: Due to the specifics of the RTF document format, Docklight cannot efficiently append new data to an existing log file, but needs to create a temporary copy of the existing log first, which can cause additional delays. It is also not supported to append new logging data with different colors & font settings than at the start of the file.

Log file directory and base name
Choose the directory and base file name for the log file(s) here. The actual file path used for the individual log file representations are displayed in the text boxes within the "Log file representation" frame.

Overwrite / append mode
Choose "append new data" if you do not want Docklight to overwrite existing log file(s). Docklight will then insert a "start logging / stop logging" message when opening / closing the log files. This is so that when in 'append mode' it is still possible to see when an individual log file session started or ended.

Representation
A separate log file may be created for each data representation (ASCII, HEX, ...). Choose at least one representation. The log files will have a ".txt" or ".htm" file extension. Docklight additionally adds the representation type to the file name to distinguish the different log files. E.g. if the user specifies "Test1" as the base log file name, the plain text ASCII log file will be named "Test1_asc.txt", whereas the plain text HEX log file will be named "Test1_hex.txt".

Disable communication window while logging
If you are monitoring a high-speed communication link or if you are running Docklight on a slow computer, Docklight may not be able to process all the transmitted data or may even freeze (no response to any user input).
Using this option to disable the communication window output while logging the data to a file. Docklight will run much faster, since the continuous display formatting and update requires considerable CPU time.

NOTE: For more information on high-speed applications, see also the section How to Increase the Processing Speed...

8.5 Dialog: Customize HTML Output

(via menu **Tools > Start Communication Logging ...** , then choose **HTML file for web browsers (.htm)** and click **Customize HTML output**)

This dialog allows you to change the appearance of the HTML log files, by modifying the HTML template code that Docklight uses when generating the HTML log file data.

You need some basic understanding of HTML documents and CSS style attributes. We recommend www.htmldog.com (English) or www.selfhtml.org (German) for a quick overview on these topics.

HTML Header Template
The HTML document header. Here you can change the font applied to the log file data, using the following CSS style attributes:

CSS Style Attribute	Description and Example

font-family	Defines one or several fonts (or: font categories) that the HTML browser should use to print a text. If the browser does not support the first font, it will try the second one, a.s.o. The last font usually defines a generic font category that every browser supports. Examples:
	`font-family:'Courier New', Courier, monospace`
	`font-family:'Times New Roman', Times, serif`
	`font-family:arial, helvetica, sans-serif`

font-size	Specifies the font size. Both, absolute and relative sizes are possible. Examples for absolute font sizes:
	`font-size:12pt`
	`font-size:xx-small`
	`font-size:x-small`
	`font-size:small`
	`font-size:medium`
	`font-size:large`
	`font-size:x-large`
	`font-size:xx-large`
	Examples for relative font sizes (relative to the parent HTML element)
	`font-size:smaller`
	`font-size:larger`
	`font-size:90%`

NOTE: Use the semicolon (";") as a separator between two different CSS style attributes, e.g.
`font-family:sans-serif; font-size:small`

NOTE: Docklight will insert additional <u> (underline), <i> (italic) and <b> (bold) HTML tags, if such formatting options are activated in the Display Settings. You do not have to use the **font-style** or **font-weight** attribute to create these effects.

HTML Footer Template
Adds additional footer text and closes the HTML document.

Data Element Template
For every new piece of log file information (channel 1 data, channel 2 data, or a comment text), a new tag with different text color is added to the HTML log file.

The template code for the header, footer and data parts contains Docklight-specific wildcards which must not be deleted:

Wildcard	Description
%BACKCOLOR%	The background color, as selected in the Display Settings
%HEADERMSG%	Header text at the start of the log file
%FOOTERMSG%	Footer text at the end of the log file

%DATA%	a chunk of the log file data: channel 1 data, channel 2 data, or a comment text
%TEXTCOLOR%	The text color to apply for %DATA%, as selected in the Display Settings

When generating a log file, Docklight replaces the wildcards with the current display settings and the actual communication data.

8.6 Dialog: Find Sequence

Menu **Edit > Find Sequence in Communication Window...**

The **Find Sequence** function searches the contents of the communication window. The search is performed in the communication window tab that is currently selected (ASCII, HEX, Decimal or Binary). You may, however, define your search string in any other representation.

Searching the communication windows is only possible if the communication is stopped.

You can search for anything that is already defined as a Send Sequence or a Receive Sequence, or you may define a custom search sequence.

NOTE: If you are looking for a sequence within the ASCII communication window, please remember the following limitations:

- The **Find Sequence** function is not able to locate sequences containing non-printing control characters (ASCII decimal code < 32) or other special characters (decimal code > 127). This is due to the nature of the ASCII display. Search using the HEX or Decimal communication window tab instead.
- In ASCII mode, the **Find Sequence** function will treat date/time stamps and any other comments in the same way as regular communication data. In HEX / Decimal / Binary mode, all additional information is ignored as long as it does not look like a character byte value.

8.7 Dialog: Send Sequence Parameter

Type in one or several value(s) for a Send Sequence with wildcards here. As with the Edit Send/Receive Sequence dialog, you may use control character shortcuts or clipboard functions.

Parameter No.
A Send Sequence can contain any number of wildcards. Each set of consecutive wildcards is considered a separate parameter. The value for each parameter is entered separately.

Minimum Characters Required
For each '?' wildcard exactly one character is required. Therefore, the minimum number of characters required is equal to the number of '?' wildcards within one parameter.

NOTE: While the Send Sequence Parameter dialog is shown, all serial communication is paused. Docklight does not receive any data and does not send any (periodical) Send Sequences.

8.8 Dialog: Project Settings - Communication

Menu **Tools >** **Project Settings... | Communication**

Communication Mode

Send/Receive
Docklight acts both as transmitter and receiver of serial data. This mode is used when Testing the functionality or the protocol implementation of a serial device or simulating a serial device.
Naming conventions: The received data (RX) will be displayed and processed as "Channel 1", the transmitted data (TX) will be displayed as "Channel 2".

Monitoring
Docklight receives serial data on two different communication channels. This mode is used, for example, when Monitoring the communication between two devices.
Naming conventions: The serial data from device 1 is "Channel 1", the data from device 2 is "Channel 2".

Communication Channels - Serial COM ports or Docklight TAP/VTP
In Docklight, a communication channel can be configured as

- Serial COM port (RS232, RS422 or RS485)
- TAP port for Docklight Tap monitoring
- VTP port for Docklight Tap Pro or Tap 485 monitoring

For serial COM port applications, choose one (or in Monitoring Mode: two) COM ports from the dropdown list. The dropdown list shows all COM ports available on your PC via the *Windows* operating system. You can also type in any COM port from COM1 to COM256 manually.

For Docklight Tap monitoring applications, open the dropdown list and choose the TAP port (e.g. **TAP0** for Channel 1, and **TAP1** for Channel 2) from the 'USB Tap' section below the COM ports. The TAP connections are only available if Communication Mode is set to 'Monitoring', the Docklight Tap is plugged in and the Docklight Tap USB device drivers are installed properly.

For Docklight Tap Pro or Tap 485 monitoring, choose VTP ports (e.g. **VTP0** / **VTP1**).

Setting / Examples	Description
COM*xxx* COM1 COM256	The channel is connected to a serial COM port. Use the dropdown list to see all COM ports available on your PC from the *Windows* operating system.
TAP*x* TAP0 TAP1	The channel is connected to one of the Docklight Tap monitoring data directions. The TAP connections are only available if Communication Mode is set to 'Monitoring', the Docklight Tap is plugged in and the Docklight Tap USB device drivers are installed properly.
VTP*x* VTP0 VTP1	The VTP*x* channel is connected to one of the Tap Pro / Tap 485 monitoring data directions, similar to the Docklight Tap application using TAP*x* settings.

COM Port Settings

Baud Rate
Choose a standard baud rate from the dropdown list, or use a non-standard baud rate by typing any integer number between 110 and 9999999.

NOTE: Non-standard baud rates may not work correctly on all COM ports, dependent on the capabilities of your COM port's hardware UART chip. You will receive no warning, if any non-standard rate cannot be applied.

NOTE: Although Docklight's Project Settings allow you to specify baud rates up to 9 MBaud, this does not mean Docklight is able to handle this level of throughput continuously. The average data throughput depends very much on your PC's performance and the Docklight display settings. See also How to Increase the Processing Speed.

NOTE: There are many COM ports drivers and applications that do not use actual RS232/422 or 485 transmission, and do not require any of the RS232 communication parameters. In some cases such COM port drivers even return an error when trying to set the RS232 parameters, so Docklight would fail to open the COM channel. Use the Baud Rate setting **None** for these applications.

Data Bits and Stop Bits
Specify the number of data bits and stop bits here. As with the baud rate, some of the available settings may not be supported by the COM port device(s) on your PC.

Tap 485 Sign. Level.
The Docklight Tap Pro / Tap 485 support additional voltage levels, besides the standard RS232 voltages:

- **RS485/422** - the differential voltage levels for RS485 and RS422 bus applications.
- **Inverted** - Inverted RS232/TTL mode, where the mark state (or logical 1) is the positive voltage, and the space state (logical 0) is the negative voltage or zero volts.

Parity
All common parity check options are available here. (The settings 'Mark' and 'Space' will probably not be used in practical applications. 'Mark' specifies that the parity bit always is 1, 'Space' that the parity bit is always 0, regardless of the character transmitted.)

Parity Error Character
This is the character that replaces an invalid character in the data stream whenever a parity error occurs. You should specify an ASCII character (printing or non-printing) that does not usually appear within your serial data stream. Characters may be defined by entering the character itself or entering its decimal ASCII code (please enter at least two digits).

NOTE: Choose "(ignore)" for the Parity Error Character if you need to transmit/receive the parity bit but Docklight should preserve all incoming characters, even when the parity bit is wrong. This is useful for applications where a 9th bit is used for addressing purposes and not for error checking.

Using Baud Rate Scan - VTP channels only

Docklight Tap Pro / Tap 485 devices offer a baud rate scan / autodetect mode. To activate baud rate scan, choose **None** from the **Baud Rate** setting and close the Project Settings dialog. Now start the communication using menu **Run >**  **Start communication (F5)**. The Docklight Tap Pro / Tap 485 now scans the communication independently in both directions. If serial data could be detected in either data direction, the most probable settings are displayed as comments in the Communication Window. They are also are noted in the communications status bar under the main toolbar.

NOTE: The accuracy of this autodetection feature depends on the actual data stream present during the scan. A continuous stream of highly random data leads to high

detection accuracy, while small transfers of individual bytes or repeating patterns may lead to wrong baud rates, data bit or parity guesses.

8.9 Dialog: Project Settings - Flow Control

Menu **Tools >** **Project Settings... | Flow Control**

Used to specify additional hardware or software flow control settings for serial communications in Docklight Send/Receive Mode.

Flow Control Support

Off
No hardware or software flow control mechanism is used. RTS and DTR are enabled when the COM port is opened.

Manual
Use this mode to control the RTS and DTR signals manually and display the current state of the CTS, DSR, DCD and RI lines. If flow control is set to "Manual", an additional status element is displayed in the Docklight main window. You may toggle the RTS and DTR lines by double clicking on the corresponding indicator.

NOTE: Flow control signals are not treated as communication data and will not be displayed in the communication window or logged to a file.

Hardware Handshaking, Software Handshaking
Support for RTS/CTS hardware flow control and XON/XOFF software flow control. These are expert settings rarely required for recent communication applications.

RS485 Transceiver Control
Some RS232-to-RS485 converters require manual RTS control, i.e. the RS232 device (PC) tells the converter when it should enable its RS485 driver for transmission. If you choose "RS485 Transceiver Control", the COM port sets RTS to High before transmitting the first character of a Send Sequence, and resets it to Low after the last character has been transmitted.

NOTE: Many USB-to-Serial converters or virtual COM port drivers do not implement the *Windows* RTS_CONTROL_TOGGLE mode properly. If you experience problems with RS485 Transceiver Control, try using a PC with an on-board COM interface or a standard PCI COM card.

8.10 Dialog: Project Settings - Communication Filter

Menu **Tools >** **Project Settings... | Communication Filter**

Contents Filter
Use this option if you do not need to see the original communication data on the serial line and only require the additional comments inserted by a Receive Sequence. This is useful for applications with high data throughput, where most of the data is irrelevant for testing and you only need to watch for very specific events. These events (and related display output) can be defined using Receive Sequences.

Channel Alias
This allows you to re-label the two Docklight data directions according to your specific use case. E.g. [Docklight] / [Device] instead of [TX] / [RX]. Or [Master] / [Slave] instead of [TAP0] / [TAP1].

8.11 Dialog: Options

Menu **Tools >** **Options...**

Display

Formatted Text Output (Rich Text Format)
used for setting the appearance of the Docklight communication window. The two different serial data streams, "Channel 1" and "Channel 2", may be displayed using different colors and styles. The standard setting uses different colors for the two channels, but using different font styles (e.g. Italics for "Channel 2") is also possible. You may also choose the overall font size here.

NOTE: If you change the font size, the communication window contents will be deleted. For all other changes, Docklight will try to preserve the display contents.

Plain Text Output (faster display, but no colors & fonts)
The formatted text output is similar to a word processor and consumes a considerable amount of CPU time. It also requires frequent memory allocation and deallocation which might decrease your PC performance. So if you are monitoring a high-speed communication link, but still want to keep an eye on the serial data transferred, try using the "Plain Text Output" format.

Control Characters (ASCII 0 - 31)
For communication data containing both printing ASCII text as well as non-printing control characters, it is sometimes helpful to see the names of the occurring control characters in the ASCII mode display window. Docklight provides an optional display settings to allow this. You can also suppress the control characters (except CR and LF) for cases when this would clutter your display.

Display Modes

Communication Window Modes
By default, Docklight will display four representations of the serial data streams: ASCII, HEX, Decimal and Binary. You may deactivate some of these modes to increase Docklight's overall performance. For example, the Binary representation of the data is rarely required. Disabling Binary mode for the communication window will considerably increase processing speed. Even when turned off for the communication window, logging in all formats is still possible.

See also the **Plain Text Output** option above.

Date/Time Stamps

Adding a Date/Time Stamp
Docklight adds a date/time stamp to all data that is transmitted or received. You may choose to insert this date/time stamp into the communication window and the log file whenever the data flow direction changes between Channel 1 and Channel 2.

For applications where the data flow direction does not change very often, you may want to have additional date/time stamps at regular time intervals. For this, activate the **Clock - additional date/time stamp...** option then and choose a time interval.

On a half duplex line (e.g. 2 wire RS485), changes in data direction are difficult to detect. Still, in most applications there will be a pause on the communication bus before a new device starts sending. Use the **Pause detection...** option to introduce additional time stamps and make the pauses visible in your communication log.

Date/Time Format
Docklight offers time stamps with a resolution of up to 1/1000 seconds (1 millisecond). For compatibility to earlier Docklight versions (V1.8 and smaller), 1/100 seconds is available, too.

NOTE: The resulting time tagging accuracy can be considerably different, e.g. 10-20 milliseconds only. The actual accuracy depends on your serial communications equipment, your PC configuration, the Docklight Display Settings (see above) and the Docklight Expert Options. See the section How to Obtain Best Timing Accuracy for details.

Control Characters Shortcuts

Here you can define your own keyboard shortcuts for ASCII Control Characters (ASCII code < 32), or for any character code > 126. Keyboard shortcuts can be used within the following Docklight dialogs and functions

- Dialog: Edit Send Sequence
- Dialog: Edit Receive Sequence
- Dialog: Find Sequence
- Dialog: Send Sequence Parameter
- Keyboard Console

For each character from decimal code 0 to 31 and from 127 to 255, you can define a keyboard combination to insert this character into a sequence (**Shortcut**). You may also define a letter which is used to display this control character when editing a sequence in ASCII mode (**Editor**).

Double click to change the value of a **Shortcut** or **Editor** field.

Predefined shortcuts are:
Ctrl+Enter for carriage return / <CR> / decimal code 13
Ctrl+Shift+Enter for line feed / <LF> / decimal code 10

8.12 Dialog: Expert Options

Menu **Tools > Expert Options...**

Expert Options are additional settings for specialized applications with additional requirements (e.g. high time tagging accuracy).

Performance

Communication Driver Mode

Use **External / High Priority Process** mode to work around a common problem for any Windows user mode application: unspecified delays and timing inaccuracies can be introduced by the Windows task/process scheduling, especially if you are running other applications besides Docklight.

External / High Priority Process mode is recommended for high accuracy / low latency monitoring using the Docklight Tap.

NOTE: For even higher and guaranteed time tagging accuracy, use the Docklight Tap Pro / Tap 485 accessories. Their accuracy does not depend on Windows and driver

latencies, and High Priority Process mode is not required for Tap Pro and Tap 485 applications.

In **External / High Priority Process** mode, the data collection in Docklight becomes a separate Windows process with "Realtime" priority class. It will be executed with higher priority than any other user application or additional application software such as Internet Security / Antivirus. For best results Docklight needs to be **Run as administrator**. Otherwise the data collection process will run with the maximum priority permitted by the OS, but not "Realtime class".

External / High Priority Process mode must be used with care, especially when you intend to monitor a high-speed data connection with large amounts of data. The PC might become unresponsive to user input. To resolve such a situation, simply "pull the plug": First disconnect the data connections or the monitoring cable to bring down the CPU load and restore the responsiveness. Then choose **Stop communication** in Docklight.

NOTE: See the section How to Obtain Best Timing Accuracy for some background information on timing accuracy.

Docklight Monitoring Mode

When Monitoring Serial Communications Between Two Devices, all received data from one COM port is re-sent on the TX channel of the opposite COM port by default ("Data Forwarding"). This is intended for special applications that require routing the serial data traffic through Docklight using standard RS232 cabling.

Use the **No Data Forwarding** Expert Option for applications with two serial COM ports where you need to avoid that any TX data is sent. This can be used to improve performance when using a Docklight Monitoring Cable, or to work around problems caused with unstable serial device drivers.

For Docklight Tap applications (e.g. using Communication Channel TAP0 / TAP1), the 'Data Forwarding' setting has no effect. The Docklight Tap is accessed in read-only mode always, and no data is forwarded.

Devices

Windows COM Devices

Use **Disable I/O error detection** when receiving repeated error messages like this:

```
DOCKLIGHT reports: General I/O error on COM1
```

NOTE: Docklight uses Windows Serial Communications in "overlapped I/O" mode for best efficiency and timing accuracy, and it continuously evaluates errors from the related Win32 API calls. In rare situations like COM devices using faulty or outdated COM device drivers, such errors can appear even in standard read/write operation. In this case, you can use this option to revert to the behavior of Docklight V2.2 and earlier versions: simply ignoring such errors.

Tap Pro / Tap 485

The firmware update functions for our Docklight Tap Pro / Docklight Tap 485 hardware accessories are only required in rare situations. E.g. if you are using an older device (< year 2017) which does not support the baud rate scan feature yet.

8.13 Keyboard Console

The Keyboard Console tool allows you to send keyboard input directly to the serial port. It can be activated using the menu **Tools > Keyboard Console On**. The keyboard console is only available for communication mode Send/Receive.

After activating the keyboard console, click in the communication window and type some characters.
Docklight will transmit the characters directly through the selected serial port. The communication window will display the characters the same way it does a Send Sequence.

NOTE: The Keyboard Console tool supports pasting and transmitting a character sequence from the clipboard, using **Ctrl + V**. This is similar to pasting clipboard data inside the Edit Send Sequence Dialog. Clipboard contents that exceeds the maximum sequence size of 1024 characters gets truncated.

NOTE: The keyboard console is not a full-featured terminal and does not support specific terminal standards, such as VT 100. The **Enter** key is transmitted as **<CR>** (ASCII 13) plus **<LF>** (ASCII 10). The **ESC** key sends **<ESC>** (ASCII 27). Use control character shortcuts to send other ASCII control characters.

NOTE: The keyboard console does not support inserting ASCII characters by pressing/holding ALT and using the numeric keypad. Please use the Edit Send Sequences dialog in HEX or Decimal representation to create any ASCII character code > 127.

8.14 Checksum Specification

Checksum specifications are used in Edit Send Sequence and Edit Receive Sequence dialogs . See Calculating and Validating Checksums for a general overview.

Supported Checksum Specifications / *checksumSpec* Argument

checksumSpec	Checksum algorithm applied
MOD256	Simple 8 bit checksum: Sum on all bytes, modulo 256.
XOR	8 bit checksum: XOR on all bytes.
CRC-7	7 bit width CRC. Used for example in MMC/SD card applications. An alternative *checksumSpec* text for the same checksum type would be: CRC:7,09,00,00,No,No See the "CRC:*width*, *polynomial...*" syntax described in the last row.
CRC-8	8 bit width CRC, e.g. for ATM Head Error Correction. Same as: CRC:8,07,00,00,No,No
CRC-DOW	8 bit width CRC known as DOW CRC or CCITT-8 CRC. Can be found in Dallas iButton(TM) applications. Same as: CRC:8,31,00,00,Yes,Yes
MOD65536	Simple 16 bit checksum: Sum on all bytes, modulo 65536.
CRC-CCITT	16 bit width CRC as designated by CCITT. Same as: CRC:16,1021,FFFF,0000,No,No
CRC-XMODEM	16 bit width CRC similar to CRC-CCITT, but the initial value is zero. Same as: CRC:16,1021,0000,0000,No,No
CRC-16	16 bit width CRC as used in IBM Bisynch, ARC. Same as:

	CRC:16,8005,0000,0000,Yes,Yes
CRC-MODBUS	16 bit width CRC as used in Modbus. Similar to CRC-16, but with a different init value. Same as: CRC:16,8005,FFFF,0000,Yes,Yes
CRC-32	32 bit CRC as used in PKZip, AUTODIN II, Ethernet, FDDI. Same as: CRC:32,04C11DB7,FFFFFFFF,FFFFFFFF,Yes,Yes
-MOD256 *or* LRC	Similar to MOD256, but returns the negative 8 bit result, so the sum of all bytes including the checksum is zero. This is equivalent to what is known as LRC (Longitudinal redundancy check) used e.g. in POS applications.
LRC-ASCII	Like -MOD256 / LRC, but it expects the source data to be HEX numbers as readable ASCII text. See the MODBUS ASCII example below.
CRC:*width*, *polynomial*, *init*, *finalXOR*, *reflectedInput*, *reflectedOutput*	Generic CRC calculator, where all CRC parameters can be set individually: *width* : The CRC width from 1..32. *polynomial* : HEX value. The truncated CRC polynomial. *init* : HEX value. The initial remainder to start off the calculation. *finalXor* : HEX value. Apply an XOR operation on the resulting remainder before returning it to the user. *reflectedInput* : Yes = Reflect the data bytes (MSB becomes LSB), before feeding them into the algorithm. *reflectedOutput* : Yes = Reflect the result after completing the algorithm. This takes places before the final XOR operation.

Remarks

Each of the predefined CRC algorithms (CRC-8, CRC-CCITT, ...) can be replaced by a specification string for the generic CRC computation (CRC:8,07,00...) as described above. We have carefully tested and cross-checked our implementations against common literature and resources as listed in the CRC Glossary.

Unfortunately there are a lot of CRC variations and algorithms around, and choosing (not to mention: understanding) the right CRC flavor can be a rather difficult job. A good way to make sure your CRC calculation makes sense is to run it over an ASCII test string of "123456789". This is the most commonly used testing string, and many specifications will refer to this string and provide you the correct checksum the CRC should return when applied on this string.

Checksums in Edit Send Sequence / Edit Receive Sequence

In the **Checksum** tab, choose one of the predefined definition strings from the drop-down list, or type in your own definition in the following format:

[(*startPos*, *len*)] ***checksumSpec*** [A *or* L] [@ *targetPos*] [# *optional user comment*]

with anything inside [ ] being an optional part.

Part	Description
checksumSpec	Required. String that specifies the checksum algorithm and its parameters, according to the checksumSpec Format table above.
(*startPos*, *len*) e.g. (1, 4)	Optional. Start and length of the character area that is used to calculate the checksum. By default everything before the checksum result is used.

A	Optional. If used, the resulting checksum value is converted into a HEX number as readable ASCII text. See the MODBUS ASCII example below.
L	Optional. Little Endian - the resulting checksum value is stored with the least significant byte (LSB) first. Default is Big Endian / MSB first.
@ *targetPos* e.g. @ -4	Optional. Specifies the first character position for storing the resulting checksum value. By default Docklight writes the checksum result to the last sequence data positions, unless you have specified "A" for ASCII result. In this case, the results is stored one character before the end, so there is still space for a "end of line" character, typically a CR as in Modbus ASCII.
# comment	You can type in a comment about this checksum specification

Remarks

startPos, *len* and *targetPos* support negative values, too, as a way to specify positions relative to the end of the sequence and not relative to the start of the sequence.
Examples:
startPos is -4 : start calculating at the 4th character from the end.
len is -1 : use everything until the end of the sequence.
targetPos is -1 : first (and only) byte of the result is stored at the last sequence character position.
targetPos is -2 : result is stored starting at the 2nd character from the end.
targetPos is -3 : result is stored starting at the 3rd character from the end.

Examples

Checksum Specification	Send Sequence Example	Actual TX Data	Remarks
`# (off, no checksum)`	01 | 02 | 03 | 04 | 05 | 00	`01 02 03 04 05 00`	after a **#** you can type in any comment to describe your checksum
`MOD256` `# simple one byte sum on all but the last character`	01 | 02 | 03 | 04 | 05 | **00**	`01 02 03 04 05 0F`	As a checksum placeholder, an extra **00** was added, but you can use any value from **00-FF**.
`CRC-MODBUS L` `# Modbus RTU checksum. Lower Byte first ('Little Endian')`	01 | 06 | 01 | 02 | 00 | 07 | **FF** | **FF**	`01 06 01 02 00 07 68 34`	**CRC-MODBUS** is a 16 bit checksum which is placed at the last two character positions in the sequence data by default.
`(2, -5) LRC-ASCII A @ -4` `# MODBUS ASCII checksum is LRC over readable HEX data, excluding start ':' and end 'CR/LF'`	: | 1 | 1 | 0 | 3 | 0 | 0 | 6 | b | 0 | 0 | 0 | 3 | **X** | **X** | r | n	`:1103006b00037E<CR><LF>`	**LRC-ASCII** treats the sequence data as a readable HEX string, where each data byte is represented by two characters. Using the **A** option produces a readable 2-letter checksum text, instead of a one character result. The **@ -4** places the result at the 4th

			character position from the right (leaving the trailing CR / LF intact).
CRC:8,07,00,00,No,Yes # CRC with custom, non-standard spec	01 | 02 | 03 | 04 | 05 | **00**	01 02 03 04 05 **3D**	Rare or custom CRCs flavors can be calculated by Docklight, but you need to know the required CRC calculation parameters. For more details see the resources listed in the CRC Glossary.

Support

9 Support

9.1 Web Support and Troubleshooting

For up-to-date FAQs and troubleshooting information, see our online support pages available at

www.docklight.de/support/

For Docklight-related news and information about free maintenance updates, see:

www.docklight.de/news.htm

For information about upgrading to Docklight Scripting (TCP, UDP, HID applications and automated testing), see:

www.docklight.de/upgrade.htm

9.2 E-Mail Support

We provide individual e-mail support to our registered customers. Please include your Docklight license key number in your request. We will contact you as soon as possible to find a solution to your problem. Send your support request to

support@docklight.de

Appendix

10 Appendix

10.1 ASCII Character Set Tables

Control Characters

Dec	Hex	ASCII Char.	Meaning
0	00	NUL	Null
1	01	SOH	Start of heading
2	02	STX	Start of text
3	03	ETX	Break/end of text
4	04	EOT	End of transmission
5	05	ENQ	Enquiry
6	06	ACK	Positive acknowledgment
7	07	BEL	Bell
8	08	BS	Backspace
9	09	HT	Horizontal tab
10	0A	LF	Line feed
11	0B	VT	Vertical tab
12	0C	FF	Form feed
13	0D	CR	Carriage return
14	0E	SO	Shift out
15	0F	SI	Shift in/XON (resume output)
16	10	DLE	Data link escape
17	11	DC1	XON - Device control character 1
18	12	DC2	Device control character 2
19	13	DC3	XOFF - Device control character 3
20	14	DC4	Device control character 4
21	15	NAK	Negative Acknowledgment
22	16	SYN	Synchronous idle
23	17	ETB	End of transmission block
24	18	CAN	Cancel
25	19	EM	End of medium
26	1A	SUB	substitute/end of file
27	1B	ESC	Escape
28	1C	FS	File separator
29	1D	GS	Group separator
30	1E	RS	Record separator
31	1F	US	Unit separator

Printing Characters

Dec	Hex	ASCII Char.	Meaning
32	20		Space
33	21	!	!
34	22	"	"
35	23	#	#
36	24	$	$
37	25	%	%
38	26	&	&
39	27	'	'
40	28	(	(
41	29	)	)

42	2A	*	*
43	2B	+	+
44	2C	,	,
45	2D	-	-
46	2E	.	.
47	2F	/	/
48	30	0	Zero
49	31	1	One
50	32	2	Two
51	33	3	Three
52	34	4	Four
53	35	5	Five
54	36	6	Six
55	37	7	Seven
56	38	8	Eight
57	39	9	Nine
58	3A	:	:
59	3B	;	;
60	3C	<	<
61	3D	=	=
62	3E	>	>
63	3F	?	?
64	40	@	@
65	41	A	A
66	42	B	B
67	43	C	C
68	44	D	D
69	45	E	E
70	46	F	F
71	47	G	G
72	48	H	H
73	49	I	I
74	4A	J	J
75	4B	K	K
76	4C	L	L
77	4D	M	M
78	4E	N	N
79	4F	O	O
80	50	P	P
81	51	Q	Q
82	52	R	R
83	53	S	S
84	54	T	T
85	55	U	U
86	56	V	V
87	57	W	W
88	58	X	X
89	59	Y	Y
90	5A	Z	Z
91	5B	[	[
92	5C	\	\
93	5D	]	]
94	5E	^	^
95	5F	_	_
96	60	`	`
97	61	a	a
98	62	b	b

99	63	c	c
100	64	d	d
101	65	e	e
102	66	f	f
103	67	g	g
104	68	h	h
105	69	i	i
106	6A	j	j
107	6B	k	k
108	6C	l	l
109	6D	m	m
110	6E	n	n
111	6F	o	o
112	70	p	p
113	71	q	q
114	72	r	r
115	73	s	s
116	74	t	t
117	75	u	u
118	76	v	v
119	77	w	w
120	78	x	x
121	79	y	y
122	7A	z	z
123	7B	{	{
124	7C	|	|
125	7D	}	}
126	7E	~	Tilde
127	7F	DEL	Delete

10.2 Hot Keys

General Hot Keys

Applies to
- Communication Window (ASCII, HEX, Decimal, Binary)
- Edit Send Sequence dialog / Edit Receive Sequence dialog
- Documentation Area

Function	Hot Key
Context-specific help	F1
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Delete	Del
Select all	Ctrl+A

Context-specific Hot Keys

Docklight menu

Menu	Function	Hot Key
File	New Project	Ctrl+N

File	Open Project	Ctrl+O
File	Save Project	Ctrl+S
File	Print Communication	Ctrl+P
Edit	Find Sequence in Comm.Window	Ctrl+F
Run	Start Communication	F5
Run	Stop Communication	F6
Tools	Start Comm. Logging	F2
Tools	Stop Comm. Logging	F3
Tools	Keyboard Console On	Ctrl+F5
Tools	Keyboard Console Off	Ctrl+F6
Tools	Minimize/Restore Documentation Area	F12
Tools	Minimize/Restore Sequence Lists	Shift+F12

Communication Window

Function	Hot Key
Find a Sequence	Ctrl+F
Clear All Communication Windows	Ctrl+W
Toggle Between ASCII, HEX, Decimal and Binary Representation	Ctrl+Tab

Send Sequences / Receive Sequences List

Function	Hot Key
Delete This Sequence	Del
Edit This Sequence	Ctrl+E
Send This Sequence *- Send Sequences List only -*	Space

Edit Send Sequence Dialog / Edit Receive Sequence Dialog

Function	Hot Key
Cancel	Esc
Wildcard '?' (matches one character)	F7
Wildcard '#' (matches one or zero characters)	F8
Function Character '&' (delay for *x* * 0.01 sec.)	F9
Function Character '%' - (Break state)	F10
Function Character '!' (handshake signals)	F11

Documentation Area

Function	Hot Key
Default Font	Ctrl+D

10.3 RS232 Connectors / Pinout

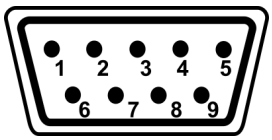
The most common connectors for RS232 communications are

- 9-pole SUB D9 (EIA/TIA 574 standard). Introduced by IBM and widely used. See below.

- 25-pole SUB D25 (RS232-C). This is the original connector introduced for the RS232 standard. It provides a secondary communication channel.
- 8-pole RJ45 (different pinouts for Cisco/Yost wiring, EIA/TIA-561, and other manufacturer-specific pinouts).

RS232 SUB D9 (D-Sub DB9) Pinout

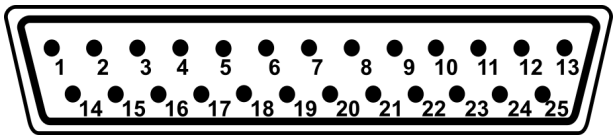
View: Looking into the male connector.
Pinout: From a DTE perspective (the DTE transmits data on the TX Transmit Data line, while the DCE receives data on this line)



Pin No.	Signal Name	Description	DTE in/out
1	DCD	Data Carrier Detect	Input
2	RX	Receive Data	Input
3	TX	Transmit Data	Output
4	DTR	Data Terminal Ready	Output
5	SGND	Signal Ground	-
6	DSR	Data Set Ready	Input
7	RTS	Request To Send	Output
8	CTS	Clear To Send	Input
9	RI	Ring Indicator	Input

RS232 SUB D25 (D-Sub DB25) Pinout

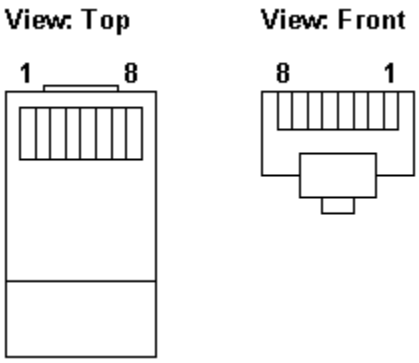
View: Looking into the male connector.
Pinout: From a DTE perspective.



Pin No.	Signal Name	Description
1	-	Protective/Shielding Ground
2	TX	Transmit Data
3	RX	Receive Data
4	RTS	Request To Send
5	CTS	Clear To Send
6	DSR	Data Set Ready
7	SGND	Signal Ground
8	DCD	Data Carrier Detect
9	-	Reserved
10	-	Reserved
11	-	Unassigned
12	SDCD	Secondary Data Carrier Detect
13	SCTS	Secondary Clear To Send
14	STx	Secondary Transmit Data

15	TxCLK	Transmit Clock
16	SRx	Secondary Receive Data
17	RxCLK	Receive Clock
18	LL	Local Loopback
19	SRTS	Secondary Request To Send
20	DTR	Data Terminal Ready
21	RL/SQ	Remote Loopback / Signal Qualify Detector
22	RI	Ring Indicator
23	CH/CI	Signal Rate Selector
24	ACLK	Auxiliary Clock
25	-	Unassigned

RJ45 8-pole pinouts



Several conflicting pinouts exist and are in use for RJ45 connectors in RS232 communications:

Cisco Console / Yost Cable / Rollover cable applications

Pinout: From a DTE perspective (the DTE transmits data on the TX Transmit Data line)

Pin No.	Signal Name	Description
1	CTS	Clear To Send
2	DCD	Data Carrier Detect
3	RX	Receive Data
4	SGND	Signal Ground
5	SGND	Signal Ground
6	TX	Transmit Data
7	DTR	Data Terminal Ready
8	RTS	Request To Send

NOTE: The Cisco/Yost pinout is used with cables that are wired "mirror image" on one end., similar to a Null Modem Cable with Handshaking. Every device has the same RJ45 female socket and transmits data on the same pin. See also the Yost Serial Device Wiring Standard .

EIA/TIA-561 standard for RJ45 / 8P8C modular connector

Pin No.	Signal Name	Description

1	DSR / RI	Data Set Ready / Ring Indicator
2	DCD	Data Carrier Detect
3	DTR	Data Terminal Ready
4	SGND	Signal Ground
5	RX	Receive Data
6	TX	Transmit Data
7	CTS	Clear To Send
8	RTS	Request To Send

NOTE: Though this is an official standard, it is more likely that you will find RS232 RJ45 products with different pinout, either the Cisco/Yost variant above or manufacturer-specific pinouts, e.g. MOXA Nport.

10.4 Standard RS232 Cables

Classic RS232 Connections

When connecting two serial devices, different cable types must be used, depending on the characteristics of the serial device and the type of communication used.

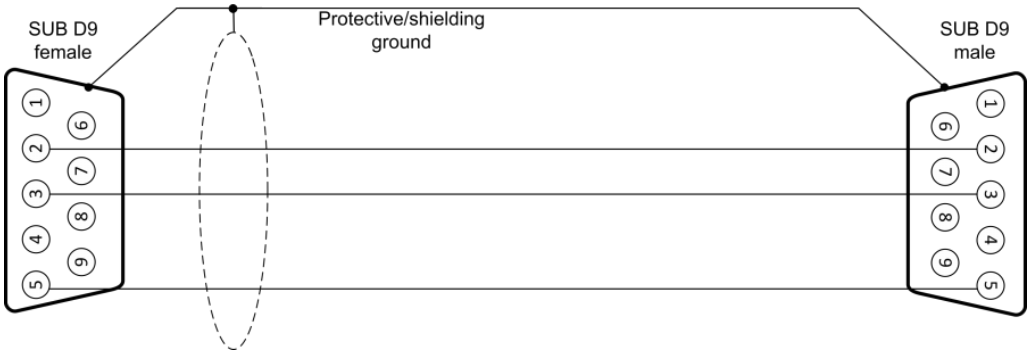
Overview of RS232 SUB D9 (D-Sub DB9) interconnections

serial device 1	serial device 2	flow control (handshaking)	recommended cable
DTE (Data Terminal Equipment)	DTE	no handshake signals	simple null modem cable
DTE	DTE	DTE/DCE compatible hardware flow control	null modem cable with partial handshaking
	DCE (Data Communications Equipment)	no handshake signals	simple straight cable
DTE	DCE	hardware flow control	full straight cable
DCE	DCE	no handshake signals	simple null modem cable, but with SUB D9 male connectors on both ends
DCE	DCE	hardware flow control	null modem cable with partial handshaking but with SUB D9 male connectors on both ends

NOTE: A great alternative to make the correct interconnection between various DTE and DCE type devices is to use the Yost Serial Device Wiring Standard approach by Dave Yost.

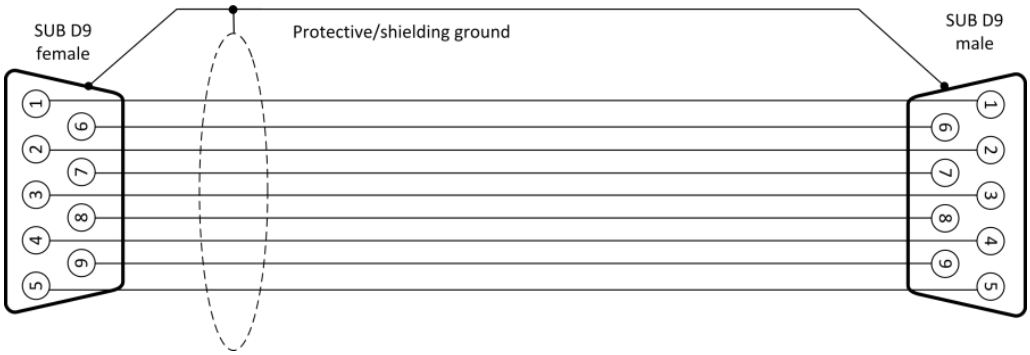
SUB D9 Simple Straight Cable

Area of Application: DTE-DCE Communication where no additional handshake signals are used.



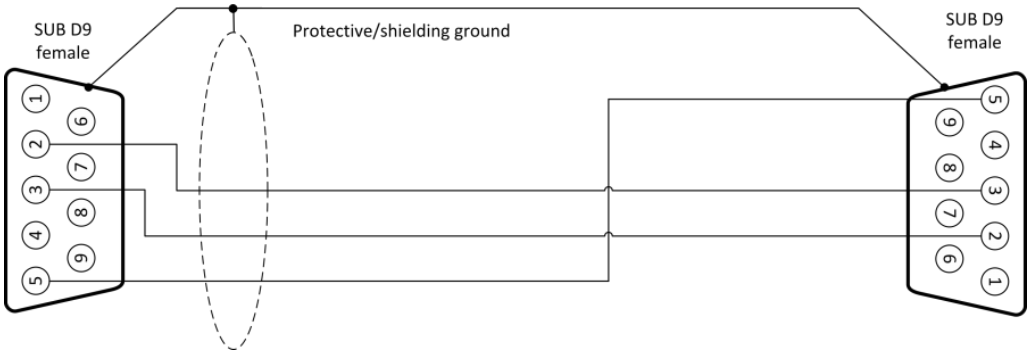
SUB D9 Full Straight Cable

Area of Application: DTE-DCE Communication with hardware flow control using additional handshake signals.



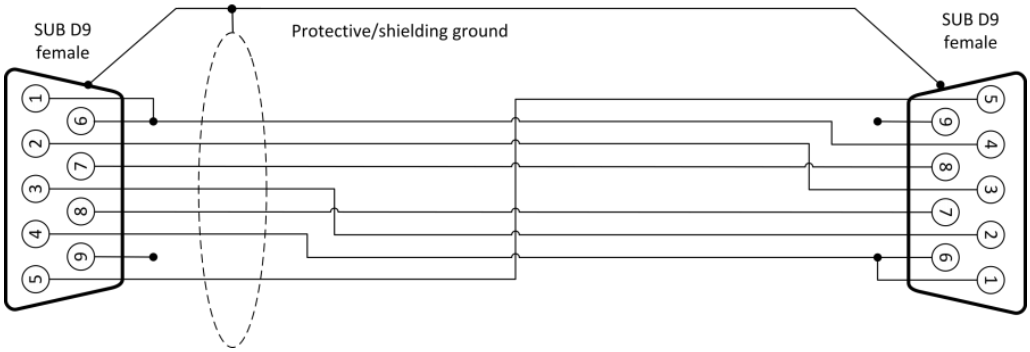
SUB D9 Simple Null Modem Cable without Handshaking

Area of Application: DTE-DTE Communication where no additional handshake signals are used.



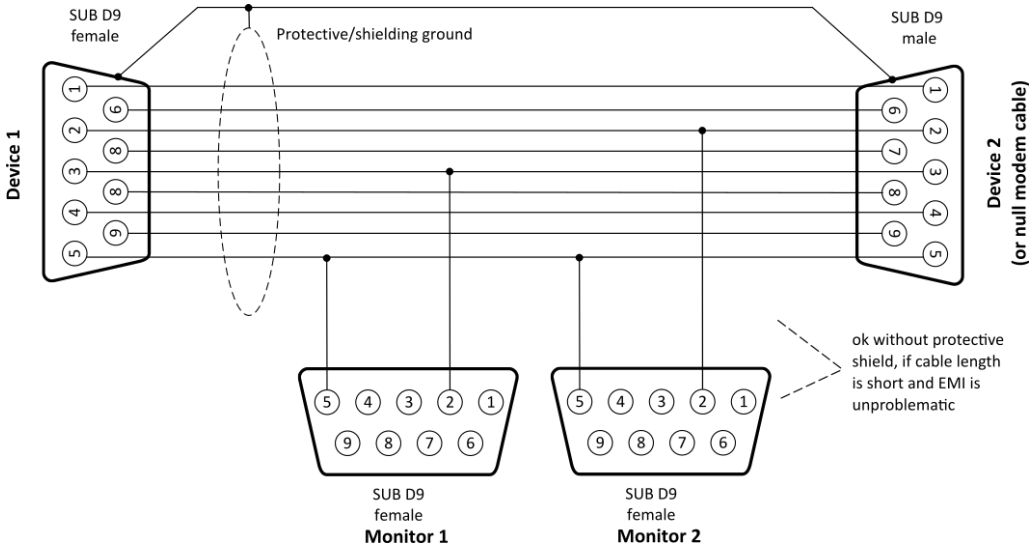
SUB D9 Null Modem Cable with Full Handshaking

Area of Application: DTE-DTE Communication with DTE/DCE compatible hardware flow control. Works also when no handshake signals are used.



10.5 Docklight Monitoring Cable RS232 SUB D9

Docklight Monitoring Cable is a RS232 full duplex monitoring cable that is designed for Monitoring serial communications between two devices.



We offer a rugged and fully shielded RS232 Monitoring cable acessory. For more details see our product overview pages and the Docklight Monitoring Cable datasheet.

NOTE: Our Docklight Tap or Tap Pro / Tap RS485 data taps offer superior monitoring characteristics, and do not require two free RS232 COM ports on your PC. Only in rare or legacy applications the Docklight Monitoring Cable is still the preferred choice today.

TIP: An inexpensive and quick solution for basic monitoring can be making your own Monitoring Cable using a flat ribbon cable and SUB D9 insulation displacement connectors, available at any electronic parts supplier.

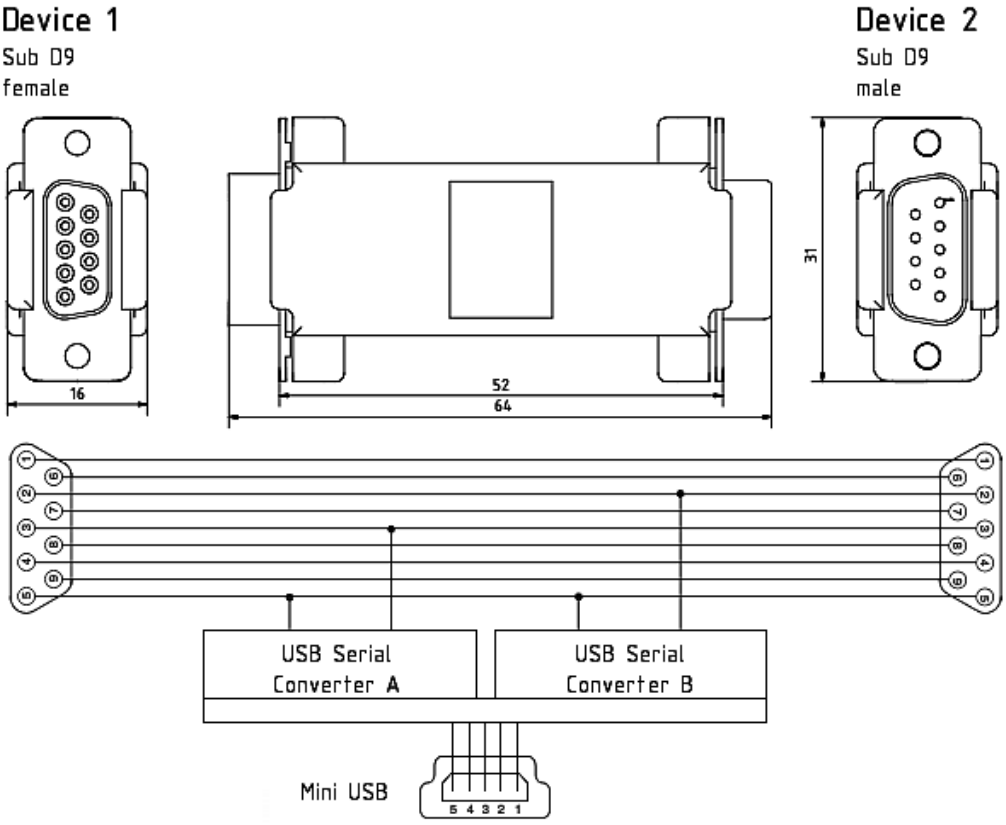
10.6 Docklight Tap

Docklight Tap is a full-duplex RS232 communications monitoring solution for the USB port.

Area of Application: Monitoring serial communications between two devices

Docklight has built-in support for the Docklight Tap. It recognizes the dual port USB serial converter and offers high-speed, low-latency access to the monitoring data. Use Docklight **Monitoring Mode** and Receive Channel settings **TAP0** / **TAP1**. See the Docklight Project Settings and How to Obtain Best Timing Accuracy for details.

Please also see our product overview pages for more information about the Docklight Tap.



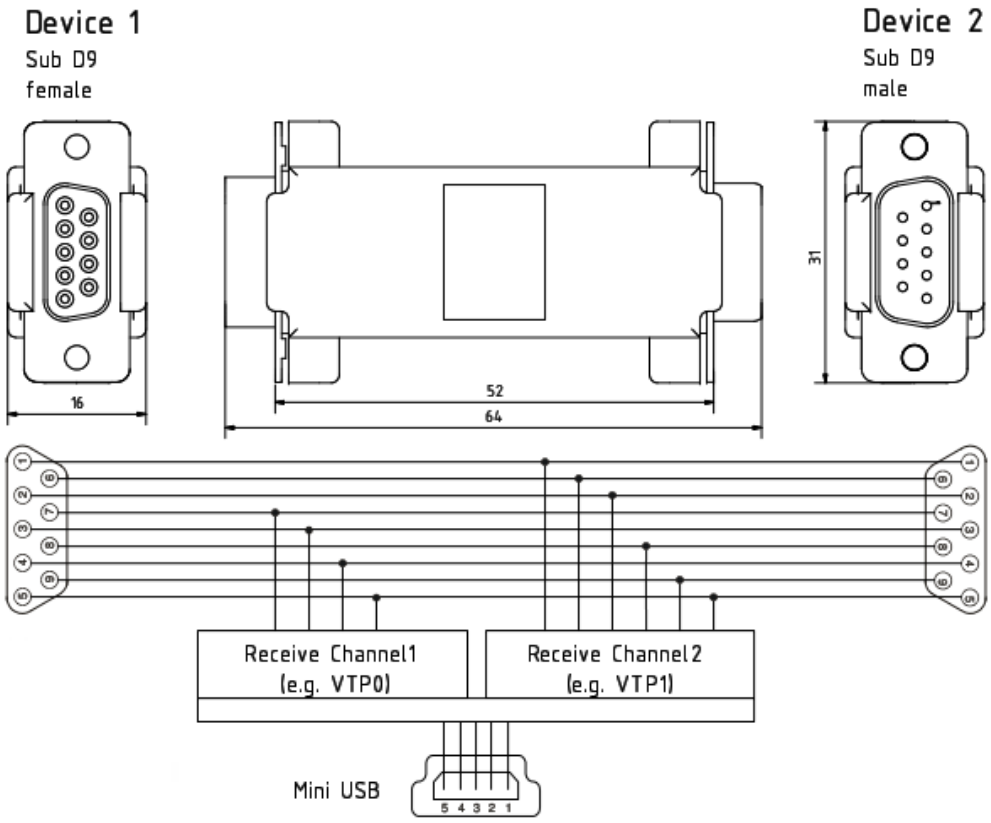
10.7 Docklight Tap Pro / Tap 485

Docklight Tap Pro and Docklight Tap 485 are advanced, high-resolution monitoring solutions for the USB port. They allow true milliseconds time measurements and monitoring high-speed data connections including RS232 status/handshake lines. They are supported by Docklight in a similar way as the Docklight Tap.

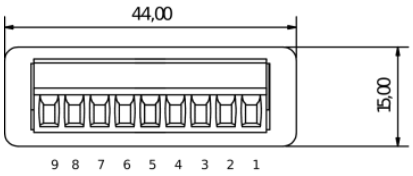
For Docklight Tap Pro and Tap 485 applications, use Docklight **Monitoring Mode** and Receive Channel settings **VTP0** / **VTP1**. See the Docklight Project Settings for more details.

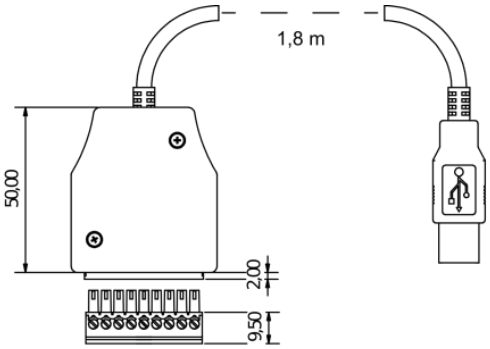
Please also see our product overview pages for more information about the Docklight Tap Pro and Docklight Tap 485.

Docklight Tap Pro



Docklight Tap RS485





inlcluding
MC 1,5 / 9-ST-3,81
Phoenix connector

USB
Type A

Glossary / Terms Used

11 Glossary / Terms Used

11.1 Action

For a Receive Sequence, the user may define an action that is performed after receiving the specified sequence. Possible actions are

- Sending a Send Sequence
  Only Send Sequences without any wildcards can be used
- Inserting a comment
  A user-defined text or an additional date/time stamp is added to the communication data window and log file
- Triggering a Snapshot
- Stopping communication

11.2 Break

A break state on an RS232 connection is characterized by the TX line going to Space (logical 0) for a longer period than the maximum character frame length including start and stop bits. Some application protocols, e.g. LIN, use this for synchronization purposes.

11.3 Character

A character is the basic unit of information processed by Docklight. Docklight always uses 8 bit characters. Nevertheless, the communication settings also allow data transmission with 7 bits or less. In this case, only a subset of the 256 possible 8 bit characters will be used but the characters will still be stored and processed using an 8 bit format.

11.4 CRC

Cyclic Redundancy Code. A CRC is a method to detect whether a received sequence/message has been corrupted, e.g. by transmission errors. This is done by constructing an additional checksum value that is a function of the message's payload data, and then appending this value to the original message. The receiver calculates the checksum from the received data and compares it to the transmitted CRC value to see if the message is unmodified. CRCs are commonly used because they allow the detection of typical transmission errors (bit errors, burst errors) with very high accuracy.

CRC algorithms are based on polynomial arithmetic, and come in many different versions. Common algorithms are CRC-CCITT, CRC-16 and CRC-32. An example of an application protocol that uses a CRC is Modbus over Serial Line.

A popular article about CRCs is "CRC Implementation Code in C" by Michael Barr, formerly published as "Slow and Steady Never Lost the Race" and "Easier Said Than Done":
https://barrgroup.com/Embedded-Systems/How-To/CRC-Calculation-C-Code

Docklight Scripting's CRC functionality (DL.CalcChecksum) was inspired by the above article and the proposed Boost CRC library:
http://www.boost.org/libs/crc/index.html

Last not least, if you are truly fascinated by CRC alchemy, you will sooner or later run into mentions of the following classic article from 1993:
"A Painless Guide to CRC Error Detection Algorithms" by Ross N. Williams:
http://ross.net/crc/crcpaper.html / http://ross.net/crc/download/crc_v3.txt

11.5 DCE

Data Communications Equipment. The terms DCE and DTE refer to the serial devices on each side of an RS232 link. A modem is a typical example of a DCE device. DCE are normally equipped with a **female SUB D9** or SUB D25 connector. See also DTE.

11.6 DTE

Data Terminal Equipment. The terms DCE and DTE refer to the serial devices on each side of an RS232 link. A PC or a terminal are examples of a typical DTE device. DTE are commonly equipped with a **male SUB D9** or SUB D25 connector. All pinout specifications are written from a DTE perspective. See also DCE.

11.7 Flow Control

Flow control provides a mechanism for suspending transmission while one device is busy or for some reason cannot further communicate. The DTE and DCE must agree on the flow control mechanism used for a communication session. There are two types of flow control: hardware and software.

Hardware Flow Control
Uses voltage signals on the RS232 status lines RTS / DTR (set by DTE) and CTS / DSR (set by DCE) to control the transmission and reception of data. See also RS232 pinout.

Software Flow Control
Uses dedicated ASCII control characters (XON / XOFF) to control data transmission. Software flow control requires text-based communication data or other data that does not contain any XON or XOFF characters.

11.8 LIN

Local Interconnect Network. A low cost serial communication bus targeted at distributed electronic systems in vehicles, especially simple components like door motors, steering wheel controls, climate sensors, etc. See also the Wikipedia entry about LIN .

11.9 Modbus

Modbus is an application layer messaging protocol that provides client/server communications between devices connected on different types of buses or networks. It is commonly used as "Modbus over Serial Line" in RS422/485 networks, but can be implemented using TCP over Ethernet as well ("Modbus TCP").

Two different serial transmission modes for Modbus are defined: "RTU mode" for 8 bit binary transmissions, and "ASCII mode". "RTU mode" is the default mode that must be implemented by all devices.

See www.modbus.org for a complete specification of the Modbus protocol.

11.10 Multidrop Bus (MDB)

Multidrop Bus (MDB) is a more exotic RS232/RS485 application, used for example in vending machine controllers, which requires a 9 bit compliant UART. The 9th data bit is used for selecting between an ADDRESS and a DATA mode.

A way to monitor and simulate such communication links using standard 8-bit UARTs, i.e. standard RS232-to-USB converters, is to use temporary parity changes.

See also Wikipedia on MDB and the original MDB 3.0 specification for more information and details.

11.11 Receive Sequence

A Receive Sequence is a sequence that can be detected by Docklight within the incoming serial data. A Receive Sequence is specified by
1. an unique name (e.g. "Modem Answer OK"),
2. a character sequence (e.g. "6F 6B 13 10" in HEX format),
3. an action that is triggered when Docklight receives the defined sequence.

11.12 RS232

The RS232 standard is defined by the EIA/TIA (Electronic Industries Alliance / Telecommunications Industry Associations). The standard defines an asynchronous serial data transfer mechanism, as well as the physical and electrical characteristics of the interface.

RS232 uses serial bit streams transmitted at a predefined baud rate. The information is separated into characters of 5 to 8 bits lengths. Additional start and stop bits are used for synchronization, and a parity bit may be included to provide a simple error detection mechanism.

The electrical interface includes unbalanced line drivers, i.e. all signals are represented by a voltage with reference to a common signal ground. RS232 defines two states for the data signals: mark state (or logical 1) and space state (or logical 0). The range of voltages for representing these states is specified as follows:

Signal State	Transmitter Voltage Range	Receiver Voltage Range
Mark (logical 1)	-15V to -5V	-25V to -3V
Space (logical 0)	+5V to +15V	+3V to +25V
Undefined	-5V to +5V	-3V to +3V

The physical characteristics of the RS232 standard are described in the section RS232 Connectors / Pinout

11.13 RS422

An RS422 communication link is a four-wire link with balanced line drivers. In a balanced differential system, one signal is transmitted using two wires (A and B). The signal state is represented by the voltage across the two wires. Although a common signal ground connection is necessary, it is not used to determine the signal state at the receiver. This results in a high immunity against EMI (electromagnetic interference) and allows cable lengths of over 1000m, depending on the cable type and baud rate.

The EIA Standard RS422-A "Electrical characteristics of balanced voltage digital interface circuits" defines the characteristics of an RS422 interface.

Transmitter and receiver characteristics according to RS422-A are:

Signal State	Transmitter Differential Voltage V_{AB}	Receiver Differential Voltage V_{AB}
Mark (or logical 1)	-6V to -2V	-6V to -200mV
Space (or logical 0)	+2V to +6V	+200mV to 6V
Undefined	-2V to +2V	-200mV to +200mV

Permitted Common Mode Voltage V_{cm} (mean voltage of A and B terminals with reference to signal ground): -7V to +7V

11.14 RS485

The RS485 standard defines a balanced two-wire transmission line, which may be shared as a bus line by up to 32 driver/receiver pairs. Many characteristics of the transmitters and receivers are the same as RS422. The main differences between RS422 and RS485 are

- Two-wire (half duplex) transmission instead of four-wire transmission
- Balanced line drivers with tristate capability. The RS485 line driver has an additional "enable" signal which is used to connect and disconnect the driver to its output terminal. The term "tristate" refers to the three different states possible at the output terminal: mark (logical 1), space (logical 0) or "disconnected"
- Extended Common Mode Voltage (V_{cm}) range from -7V to +12V.

The EIA Standard RS485 "Standard for electrical characteristics of generators and receivers for use in balanced digital multipoint systems" defines the characteristics of an RS485 system.

11.15 Send Sequence

A Send Sequence is a sequence that can be sent by Docklight. A Send Sequence is specified by
1. an unique name (e.g. "Set modem speaker volume"),
2. a character sequence (e.g. "41 54 4C 0D 0A" in HEX format).

There are two ways to make Docklight send a sequence:

- Sending a sequence can be triggered manually by pressing the send button in the Send Sequences list
  (see Main Window).
- Sending a sequence may be one possible reaction when Docklight detects a specific Receive Sequence within the incoming data (see Action).

11.16 Sequence

A sequence consists of one or more 8 bit characters. A sequence can be any part of the serial communications you are analyzing. It can consist of printable ASCII characters, but may also include every non-printable character between 0 and 255 decimal.
Example:

ATL2 (ASCII format)
41 54 4C 0D 0A (HEX format)
This sequence is a modem command to set the speaker volume on AT compatible modems. It includes a Carriage Return (0D) and a Line Feed (0A) character at the end of the line.

The maximum sequence size in Docklight is 1024 characters.

11.17 Sequence Index

The Sequence Index is the element number of a Send Sequence within the Send Sequence List, or of a Receive Sequence within the Receive Sequence List. The Sequence Index is displayed in the upper left corner of the Edit Send Sequence or Edit Receive Sequence dialog.

11.18 Serial Device Server

A Serial Device Server is a network device that offers one or more serial COM ports (RS232, RS422/485) and transmits/receives the serial data over an Ethernet network. Serial Device Servers are a common way for upgrading existing devices that are controlled via serial port and make them "network-enabled".

11.19 Snapshot

Creating a snapshot in Docklight means generating a display of the serial communication shortly before and after a Trigger sequence has been detected. This is useful when testing for a rare error which is characterized by a specific sequence. See Catching a specific sequence and taking a snapshot... for more information.

11.20 Trigger

A Trigger is a Receive Sequence with the "Trigger" option enabled (see Dialog: Edit Receive Sequence). When the Snapshot function is enabled, Docklight will not produce any output until a trigger sequence has been detected in the serial communication data. See Catching a specific sequence and taking a snapshot... for more information.

11.21 UART

Universal Asynchronous Receiver / Transmitter. The UART is the hardware component that performs the main serial communications tasks:
- converting characters into a serial bit stream
- adding start / stop / parity bits, and checking for parity errors on the receiver side
- all tasks related to timing, baud rates and synchronization

Common UARTs are compatible with the 16550A UART. They include a 16 byte buffer for incoming data (RX FiFo), and a 16 byte buffer for outgoing data (TX FiFo). Usually these buffers can be disabled/enabled using the *Windows* Device Manager and opening the property page for the appropriate COM port (e.g. COM1).

11.22 Virtual Null Modem

A virtual null modem is a PC software driver which emulates two serial COM ports that are connected by a null modem cable. If one PC application sends data on one virtual COM port, a second PC application can receive this data on the second virtual COM port and vice versa.

By using a virtual null modem driver on your PC you can easily debug and simulate serial data connections without the use of real RS232 ports and cables.

Virtual COM connections do not give you the same timing as real RS232 connections and usually do not emulate the actual bit-by-bit transmission using a predefined baud rate. Any data packet sent on the first COM port will appear in the second COM port's receive buffer almost immediately. For most debugging and simulation purposes, this limitation can be easily tolerated. Some virtual null modem drivers offer an additional baud rate emulation mode, where the data transfer is delayed to emulate a real RS232 connection and its limited transmission rate.

For an Open Source *Windows* solution that has been successfully tested with Docklight, see
https://sourceforge.net/projects/com0com/
We recommend the com0com v2.2.2.0 signed x64 version, which we tested successfully *Windows 10* and *Windows 11*:
https://sourceforge.net/projects/com0com/files/com0com/2.2.2.0/com0com-2.2.2.0-x64-fre-signed.zip/download

11.23 Wildcard

A wildcard is a special character that serves as a placeholder within a sequence. It may be used for Receive Sequences when parts of the received data are unspecified, e.g. measurement readings reported by a serial device. Wildcards can also be used to support parameters in a Send Sequence.

The following types of wildcards are available in Docklight:
Wildcard '?' (F7): Matches exactly one arbitrary character (any ASCII code between 0 and 255)
Wildcard '#' (F8): Matches zero or one character. This is useful for supporting variable length command arguments (e.g. a status word) in Send / Receive Sequences. See Checking for sequences with random characters or Sending commands with parameters for examples and additional information.
Other placeholders that allow random data:
Function Character '!' (F12): Bitwise comparison. This is useful if there are one or several bits within a character which should be tested for a certain value. See Function character '^' (F12) - bitwise comparisons for details and an example.