Table of Contents

Introduction

Device Monitoring Studio is an application with rich device monitoring capabilities. It allows you to monitor the different kind of devices attached to local or remote computer. It supports monitoring the serial ports and devices (built-in, PnP and virtual), Universal Serial Bus (USB) devices and network connections. In addition, it makes monitoring the connection between two serial devices possible by creating a virtual bridge between them (serial bridging).

Organization

This documentation is divided into the following sections:

Monitoring Session Management

This section briefly describes Device Monitoring Studio user interface elements that help you manage devices, start, stop and configure monitoring sessions.

Device Types

This section describes all supported device types, including physical (such as serial, USB or network) or virtual (like Serial Bridge and Playback) devices.

Data Processing

This section provides detailed documentation for every data processing module. Some of the modules are specific to one or more device types, while others support all device types.

Filtering

This section describes the advanced filtering capabilities provided by Device Monitoring Studio.

Advanced Features

This section provides descriptions for additional functionality available for each DMS module.

User Interface

User Interface section gives detailed description of each user interface element.

Configuration

Configuration section provides help for using centralized Device Monitoring Studio configuration mechanism, which is opened by using the **Tools » Settings...** command.

What's New in 9.12

Starting from version 9.12, Device Monitoring Studio supports monitoring of remote serial and USB devices[1].

A separate package, called Device Monitoring Studio Server needs to be installed on a remote computer. In addition, the server is also included as an optional component in Device Monitoring Studio installer. It provides with a fast and simple way to share local serial and USB devices and allow a remote Device Monitoring Studio instance to monitor them.

Remote monitoring sessions provide the same set of features as local monitoring sessions.

---

1. This feature is not available in all product editions.↩

What's New in 9.01

This is a major version update. This topic lists new key features as well as improvements and updates in existing Device Monitoring Studio features.

UI Refresh

Version 9 comes with a revamped user interface that offers a sleek and modern look. These improvements to the user interface make Device Monitoring Studio more visually appealing, user-friendly, and adaptable to various device configurations and preferences. Here are the key improvements:

New beautiful vector icon set

The new icon set is designed to be crisp and clear on any screen resolution, ensuring a consistent and visually appealing experience across all devices.

Full support for high-DPI monitors

The software now automatically adjusts its interface elements to provide optimal clarity and readability on high-resolution displays.

Support for color themes for all interface elements

Users can now customize the look and feel of Device Monitoring Studio by adjusting colors of any visual element. This allows them to personalize their experience and make the software more visually appealing.

Dark theme support

The updated version of Device Monitoring Studio offers full support for Windows 10/11 dark theme, providing a more comfortable and energy-efficient experience for users, especially in low-light environments. This feature helps reduce eye strain and improves visibility of interface elements.

Resizable windows

The new version of Device Monitoring Studio allows users to resize most dialog boxes according to their preferences. This feature provides greater flexibility and customization, allowing users to adjust the interface to suit their needs. In addition, some dialog boxes have been redesigned to improve the user experience, making them more intuitive and easier to navigate.

Localization support

Device Monitoring Studio now fully supports localization. Initial release comes with support for English, Spanish, German, French, Italian and Russian languages. New language packs may be created with a free Language Editor.

New and improved Data Visualizers

Almost all existing data visualizers have been improved in a number of ways, with better functionality and look. A number of new data visualizers (all based on Custom View technology) have been added, including USB Mass Storage View, Still Image View, Communication View, Audio View and Video View.

ARM64 support

Device Monitoring Studio now offers full support for ARM64 architecture. This means that the software can now be seamlessly run on devices powered by ARM64 processors, ensuring optimal performance and compatibility. With this new feature, users can enjoy the benefits of using it on a wider range of devices

Settings Manager

New version introduces a new Settings Manager with centralized storage for all application settings. By default, these settings are stored in the Registry, but they can be configured to be stored in a file in the file system. This file can be placed in a shared folder or OneDrive or a similar folder. This new feature allows users to easily manage and access their settings from any device, making it more convenient for them to use Device Monitoring Studio from different devices.

Walkthroughs and tutorials for new users

Device Monitoring Studio now shows a few brief walkthroughs and tutorials for novice users. Experienced users may easily turn this off.

Breaking Changes

- This version drops support for Windows 7, Windows 8 and Windows 8.1 operating systems (and corresponding server versions). It also drops support for 32-bit OS versions. Only 64-bit (x86-64 and ARM64) Windows 10 or later versions are now supported.
- DMSLOG7 log file format is no longer supported. Device Monitoring Studio 9.x does not save or load these log files.

What's New in 8.33

This release improves Structure View data visualizer. In previous versions, packet fields were not displayed until the user expanded the packet in the Structure View data visualizer. Starting from version 8.33, Structure View uses its standard field visualization algorithm to display top-level packet data. This includes, for example, the evaluation of the display attribute for a top-level or sub-level protocol.

Additionally, the new built-in visualize() function is available to protocol definition code. It provides a way to invoke standard field visualization algorithm for the protocol definition code.

Network and serial protocols have been updated to take advantage of the new feature. Now the user will be able to see the type and basic fields of each network or serial packet without expanding it in the Structure View.

What's New in 8.30

This release introduces the updated Data Recording component. New logging format allows the user to specify optional size or duration limits and split the resulting log file into several part files.

This release also brings an improved Playback Settings window.

What's New in 8.04

This release introduces the new component: HID Send, which may be used to directly control the HID device by querying its parameters and sending reports. The new component may also be fully controlled from scripts.

What's New in 8.02

Script Debugger

Increased support for scripting in Device Monitoring Studio calls for better tools. The built-in script editor with auto-completion support and built-in documentation is now accompanied by a script debugger.

The user may now place breakpoints, and perform stepping while debugging his scripts. The Watch tool window and Stack Trace tool window provide the detailed information about the script execution state.

See the Script Debugger section for more information.

New Serial Terminal

The Serial Terminal module has been redesigned and rewritten from scratch. It now provides richer API, better performance and stability and integrates with a Serial Source. This integration allows for “listening” mode monitoring sessions - a configuration when there is no active controlling application listening on the port.

In addition, terminal window is now integrated better into the application frame, is more responsive and provide better overall experience.

Data Repeater

This component, which may be attached to a serial or playback monitoring session, allows redirection of monitored traffic to one or more serial ports.

See the Data Repeater section for more information.

Improved USB Descriptors Retrieval

Device Monitoring Studio now uses alternative methods to retrieve USB device descriptors if standard methods fail.

Custom View Visualizer

The Custom View data visualizer is now out of beta. It has been redesigned and is now backed by TypeScript (not internal protocol definition language). Serial's Request View and Console View have been rewritten using Custom View.

Custom View data visualizer allows the user to consume the information produced by protocol binding module and display parsed data with rich visualization capabilities.

Extended Scripting API

This release extends the list of components that may be controlled from TypeScript (JavaScript) scripts executing inside the Device Monitoring Studio:

- File Manager object.
- TCP Session, UDP Session and TCP Listener objects.
- Updated Serial Terminal object.
- New MODBUS Builder Object.

What's New in 7.74

This release adds three new directives to the protocol definition language: $revert_to, $shift_by and $remove_to, which allows having more advanced look-ahead in protocol definitions.

It also adds USB descriptor parsing to Structure View data visualizer and Text Exporter data processing module. New predefined methods are added to USB: usb_get_vendor_name and usb_get_model_name.

What's New in 7.70

This release brings the following:

1. Starting from this version, the minimum supported OS is Windows 7. Application no longer supports Windows XP and Windows Vista. The minimum supported server OS is Windows Server 2008 R2.

2. TypeScript is now always supported, the requirement to have Internet Explorer 11 installed is removed.

3. New script editor is introduced with support for auto-completion and error highlighting.

4. Custom serial session type is implemented.

5. User interface elements now look correctly on high-definition displays.

6. Updated scripting engine with support for ES6. This allowed us to improve various APIs provided by Device Monitoring Studio components. For example, all methods that worked with Array objects now also work with typed arrays, array buffers and data views.

What's New in 7.51

This release is dedicated mostly to polishing various application features. It improves a lot of existing components by fixing bugs and memory leaks. It also significantly reduces application memory requirements and works around memory exhaustion issues, particularly on 32-bit systems.

New Features

This release introduces high-precision session time measurement mode. As of 7.51, this mode is supported by USB, Serial and Network modules. When enabled (per-session), Device Monitoring Studio uses high-precision system timer for packet time-stamping. This mode is enabled by default if Device Monitoring Studio is installed on Windows 7 or later.

Starting from 7.51, various device information is automatically saved to a log file if the Data Recording module is added to a monitoring session. This includes the Serial Device Information for serial devices and all kinds of USB descriptors for USB devices (Device Descriptor, Configuration Descriptor and HID Descriptor tool windows).

Raw Data View data visualizer now supports additional packet coloring mode. Now it allows the user to choose one of three modes: no coloring, packet interlacing (coloring odd and even packets) and read/write coloring (coloring read and write packets).

When resulting log file is later selected in the Devices tool window, corresponding tool windows are updated with saved information.

In addition, built-in MODBUS protocol now automatically selects RTU or ASCII mode. This works automatically in Structure View and in filters and does not affect MODBUS View data visualizer.

Fixed Bugs

This release fixes over 200 different bugs, including crashes. A special attention has been paid to the following:

- Monitoring session startup and shutdown-related issues, including crashes and hangs. This includes configuring and starting monitoring session, adding or removing data processing modules to running session and stopping monitoring session, as well as changing session's Capture Filter or visualizers' Display Filter for a running session.
- Capture and display filter performance.
- Protocol binding performance.
- Issues with various data visualizers. Almost every data visualizer has been optimized or fixed in some way.
- Workspace saving and loading issues.

Here is a short list of most remarkable fixed bugs:

Line View

Line View had a bug that prevented it from working. Now Line View works correctly when monitoring serial modems.

Correct emergency session termination

When resources are low, Device Monitoring Studio is now more robust in stopping affected high-data rate monitoring sessions, or session processing modules. This mostly affects 32-bit systems or systems with slow secondary storage.

Log file repairing

Device Monitoring Studio has always repaired damaged log files on startup. If log file was large, it introduced a startup delay (only splash screen was visible and application appeared to be hanging). Now a progress bar is shown to tell the user that application is alive.

Serial communication mode

A number of various bugs related to support for serial communication mode in Serial and Serial Bridge modules have been fixed. This includes saving the mode into the workspace or log file.

Reading USB device descriptors

It was impossible to read USB device descriptors on some systems. The problem seems to be related to incorrect system configuration, damaged host controller driver installation or various security issues. Device Monitoring Studio now has an alternative way to read descriptors if previous method fails. This improves monitoring experience for USB devices on affected systems.

Operating with limited processing resources

During monitoring Device Monitoring Studio uses available disk space as configured in the Data Processing Tab (invoked by **Tools » Settings** menu command). We have fixed a number of bugs related to Device Monitoring Studio operation under limited settings (like 1% on a single disk and 0% on all others).

"Exporter" data processing modules: Raw Exporter, Text Exporter and Data Recording

A number of bugs have been fixed in exporter modules. CPU and memory usage have been carefully profiled and optimized.

Memory Usage Optimizations

This release also brings improvements to Device Monitoring Studio memory usage strategies. All critical paths have been carefully analyzed and profiled, resulting in memory usage savings of up to 10 times in several places. Application is now much more robust on systems with lower amount of RAM. In addition, when installed on 32-bit operating systems, application is now capable of sustaining much higher data rates during monitoring. Reduced memory demands now also allow to start "heavier" monitoring sessions - sessions with more data processing modules, complex custom protocols and filter expressions.

Supported OSes

This release drops support for 64-bit Windows XP and Windows Server 2003. 32-bit versions of these operating systems are still supported.

Remote Source now requires at least Windows Vista and will not be installed on Windows XP or Windows Server 2003.

Future releases will drop support for Windows XP and Windows Server 2003 entirely.

What's New in 7.25

Remote Monitoring

Starting from version 7.25, active remote sessions are no longer automatically terminated when connection to remote server breaks. Instead, Device Monitoring Studio starts trying to automatically reconnect to remote server and resume monitoring.

New global object is available for scripting: Remote Connection Manager Object.

New parameter (remote server name) has been added to IHost.createSession and to ISession.addDevice function overloads. Therefore, it is now possible to start remote monitoring sessions from scripting code.

Bridge Manager Scripting Object

Bridge Manager object is now available. It allows user code to create and manage serial bridges through its methods and properties. Created bridges may be manipulated through the IBridge interface.

In addition, starting from version 7.25, it is possible to start serial bridge monitoring sessions from scripting code.

Updates to Serial Terminal Scripting

Serial Terminal Session Object adds the following property: ITerminalSession.flowControl to query or set terminal session's flow control mode.

IFlowControl interface and Predefined Flow Control Object have been added.

Updated Typescript Version

This version updates the included TypeScript compiler to version 1.5.

What's New in 7.17

Version 7.17 adds support for MODBUS TCP protocol. There is a protocol definition file for it and support for building and sending MODBUS TCP packets using MODBUS Send.

In addition, the following two objects are available for user scripts: TCP Manager Object and TCP Session Object. TCP Session Object may be used instead of Serial Terminal Session Object in a call to **IModbusManager.createSession** to create a **MODBUS Send Session Object** over MODBUS TCP protocol. (Deprecated).

Updated Typescript Version

Version 7.13 introduced the ability to write user scripts in TypeScript language (superset of JavaScript) with strong type checking, parameter validation and so on. This version also updates the included TypeScript compiler to version 1.4.

What's New in 7.13

Version 7.13 greatly improves built-in Scripting support. Starting from this version, scripting support is available for all modules and new scripting object, Monitoring Object, is introduced. By calling methods of this object, user scripts may create, configure and start monitoring sessions.

In addition, a convenient script file editor is added to Device Monitoring Studio with syntax coloring, Undo/Redo support and advanced error reporting.

From this version, user scripts may now be written in TypeScript, while support for VBScript has been discontinued. TypeScript usage provides better error checking and parameter validation.

Serial Device Parameters

Support for displaying current session parameters, such as Baud Rate, Data Bits, Stop Bits and Parity has been added to Sessions Tool Window for serial sessions.

What's New in 7.05

Technical Features and Improvements

Multi-source Monitoring

New version supports joining monitored data from multiple sources of the same type into a single monitoring session. Now it is possible to monitor two or more serial ports, USB devices or network adapters. Device Monitoring Studio makes sure packets are correctly sorted and presented through

a number of supported data visualizers. Data logging also supports multi-source sessions.

Remote Monitoring

DMS 7 supports monitoring USB and serial devices connected to remote servers. A single client may monitor several servers and a single server may be monitored by multiple clients. A separate server installation is provided. It includes a server access module, serial and USB monitoring modules, administration and management module and documentation. A server is managed using the MMC Snap-In or from Windows Scripting Host or PowerShell.

Windows 8 Support

DMS 7 officially supports Windows 8 and Windows Server 2012.

USB 3.0 Support

USB Monitor introduces support for USB 3.0 host controllers and devices.

Built-In and Custom Protocol Parsing

DMS 7 extends protocol parsing support for all monitoring modules: network, USB and serial. In addition, this version has new implementation of protocol parsing, which is more flexible and greatly improves performance.

New Data Processing Category

New processing category is introduced: data exporters. These components process monitored data in some way, but do not produce any visible output on the screen. Data recording module now belongs to this category. Other data exporters are Raw Exporter and Text Exporter. Both these exporters are capable of using built-in or custom protocol definitions to parse monitoring data before exporting.

Parallel Processing

New version of Device Monitoring Studio utilizes multiple cores for more responsive monitoring session operation. It allows the user to perform real-time monitoring of 1 Gb network transfer without packet loss and slowdowns while having such “heavy” processing like Capture Filter and Display Filter configured for a monitoring session.

User Interface Improvements

Devices Tool Window

New tool window that displays all devices the user can monitor in a single customizable view. For each supported device, its image, type and current state are displayed. The user may start, stop and configure monitoring sessions, view device properties, restart and rename devices.

This tool window allows you to create and configure Serial Bridges. Device Monitoring Studio also puts log files into corresponding places in this device tree.

Session Configuration Window

New Session Configuration window greatly simplifies session creation and modification. There is a list of configured sources at the top of the window. Below are optional device configuration settings, buttons to add more devices and remove existing ones.

Depending on the type and number of configured sources, a list of available processing modules is populated. It is divided into two main categories: visualizers and exporters.

Double-click on the processing module to add it to the current session. Some of modules support configuration. Customized processing modules may be saved for future use.

Finally, the user specifies a capture filter, a generic (protocol-based) conditional expression used to filter out specific monitored packets.

Sessions Tool Window

New tool window lists all currently running sessions and their properties. It allows you to close data visualizers, add new data processing modules, pause, resume or stop sessions. It also allows you to

change data processing modules configuration conveniently. For example, for data recording, the current log file size is displayed. By clicking “End Stream”, the user ends the current stream and starts a new one.

Protocol-Based Data Visualizers

Structure View data visualizer for USB, Serial and Network with additional filtering (Display Filter) and ability to specify root protocol.

Automatic Layout Loading

DMS 7 automatically loads separate tool window and command layout when monitoring session is started. Thus, until a session is started, a layout, which simplifies device discovery and information retrieval is used. After the session is started, another layout is automatically loaded streamlining monitoring session usage. Both layouts may be customized by the user.

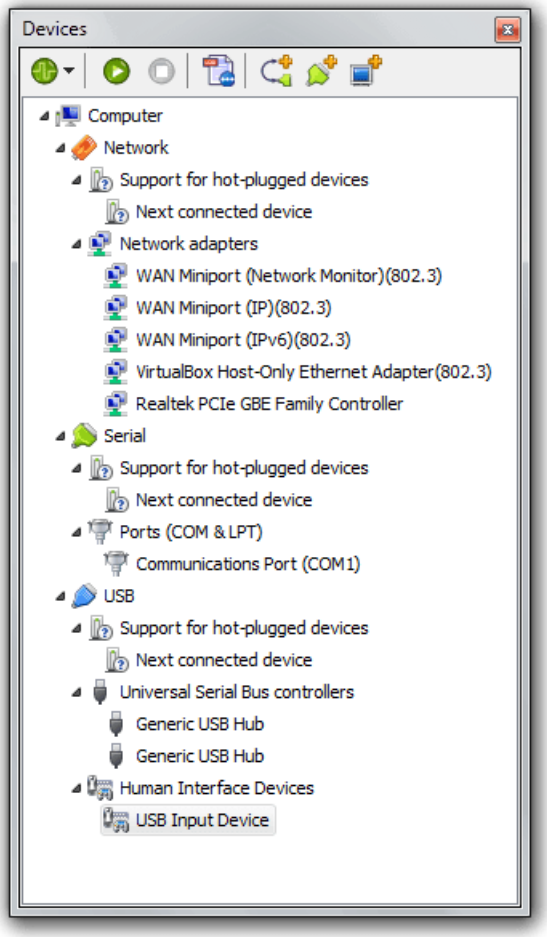
Global Configuration Switch

Device Monitoring Studio provides you with a global switch to turn it into Serial Monitor, USB Monitor or Network Monitor. This switch may be used in combo installations to temporary hide unneeded components.

Monitoring Session Management

Devices Tool Window

This tool window is a central location where all supported devices are found.



It has three levels:

1. Computer. This can be either “My Computer” or the name or address of a remote server.
2. Device type. One of three supported device types: Network, Serial or USB. Some of these types may be missing if corresponding monitoring module is not installed.
3. Device category. Category further splits the device type into sub-types. For example, there may be “Human Interface Devices” category for USB or “Ports” category for Serial.

Supported devices are listed below the *Device Category* level. This includes physical devices, virtual devices such as “Next connected device” or serial bridge, discovered log files and so on.

You may use mouse or keyboard to select a single device. This essentially makes it the “current” device and causes any compatible tool window to update its contents. For example, if you click on a “USB Input Device”, Device Descriptor tool window will update to show you USB device descriptor of the selected device. Each device type usually updates only one subset of informational tool windows.

Detailed information is provided in the corresponding documentation section.

Devices that are currently being monitored are marked with an animated gear.

Devices Tool Window automatically updates the list of devices whenever new supported device is connected to the computer or an existing device is disconnected from it.

In addition to displaying the list of supported devices, Devices tool window provides you with following features:

Commands

This section describes commands on the Devices Tool Window Toolbar. Note that some of the commands may be missing in your installation, depending on the list of installed modules.

Show or Hide Categories

Allows you to turn displaying of different device types on or off. Network, USB and Serial are main filter categories, while Playback and Serial Bridge are sub-categories. Turning the main category off hides all devices of this category, including the ones belonging to one of the sub-categories. Thus, if you turn Serial off, it will hide all serial devices, including bridges and serial log files. However, if you turn Serial Bridge category off without turning Serial category off, it will only hide serial bridges.

Start Monitoring...

Opens the Session Configuration window and allows you to start new monitoring session. Double-clicking on a device in a tree does the same.

Stop Monitoring

Closes the monitoring session to which the currently selected device belongs.

Configure Playback Path...

Opens the corresponding settings page where you can configure default log file folder location.

Create New Bridge...

Starts creation of new serial bridge.

Create Virtual Serial Port...

Opens HHD Software Free Virtual Serial Ports application that allows you to create different kinds of virtual serial ports. These ports are fully supported by Device Monitoring Studio.

Connect to New Server...

Allows you to connect to Device Monitoring Studio Server running on another computer. This effectively "adds" all supported devices on the server to the tree and allows you to monitor them remotely.

Context Menu

In addition to generic commands, each device type and sometimes category may provide you with extended list of commands. Right-click on the item in a device tree to bring up the list of supported commands.

The **Rename...** command, supported for most of the devices allows you to specify a custom name for a device. Use this ability to distinguish between similar devices. For example, a mouse and UPS both may be detected as "Human Interface Device". Use the **Rename** command to name them "Mouse" and "UPS" correspondingly.

Device Monitoring Studio remembers names you give and automatically uses them next time you start the application or reconnect devices.

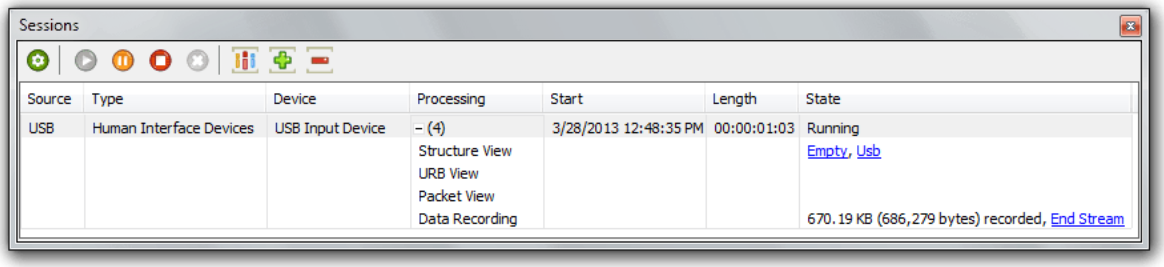
Properties... command, also present on most devices, opens the Windows Device Manager's Device Properties window, allowing you to view and sometime change device properties.

Restart Device command emulates the process of disconnecting a device and then re-connecting it to the computer. Note that this operation is not very reliable and may be unsupported by some device classes. Using the command on unsupported device may eventually require the user to physically disconnect and re-connect it.

Serial Bridge, Playback and Remote Source all add additional commands to devices and/or categories they create and maintain in the tree.

Sessions Tool Window

This tool window lists all currently running sessions and provides sessions management.



For each session, the following information is displayed:

Source

Device type (Serial, USB or Network) and remote computer name. If device is local, only device type is displayed.

Type

Device category, like “Human Interface Devices”.

Device

The name of the device this session monitors. For multisource sessions, a list of names is displayed.

Processing

A current list of processing modules configured for this session.

Start

Session start time.

Length

Session running time.

State

Depends on the specific data processing module. For example, for data recording, the total number of recorded bytes as well as commands to end current stream and start new stream are present.

The window has a following list of commands:

Configure Session...

Opens the Session Configuration window that allows you to change the session parameters (including Capture Filter) and add or remove data processing modules.

Resume Session

Resumes the paused session.

Pause Session

Effectively pauses the entire session, including all configured data processors. No monitoring activity occurs for the session while it is paused.

Stop Session

Closes all data processing modules, disconnects from the session device(s) and removes the session from the session list.

Close Visualizer

Closes selected data visualizer.

Configure Columns

Allows you to change the list of visible columns and their order.

Expand Item

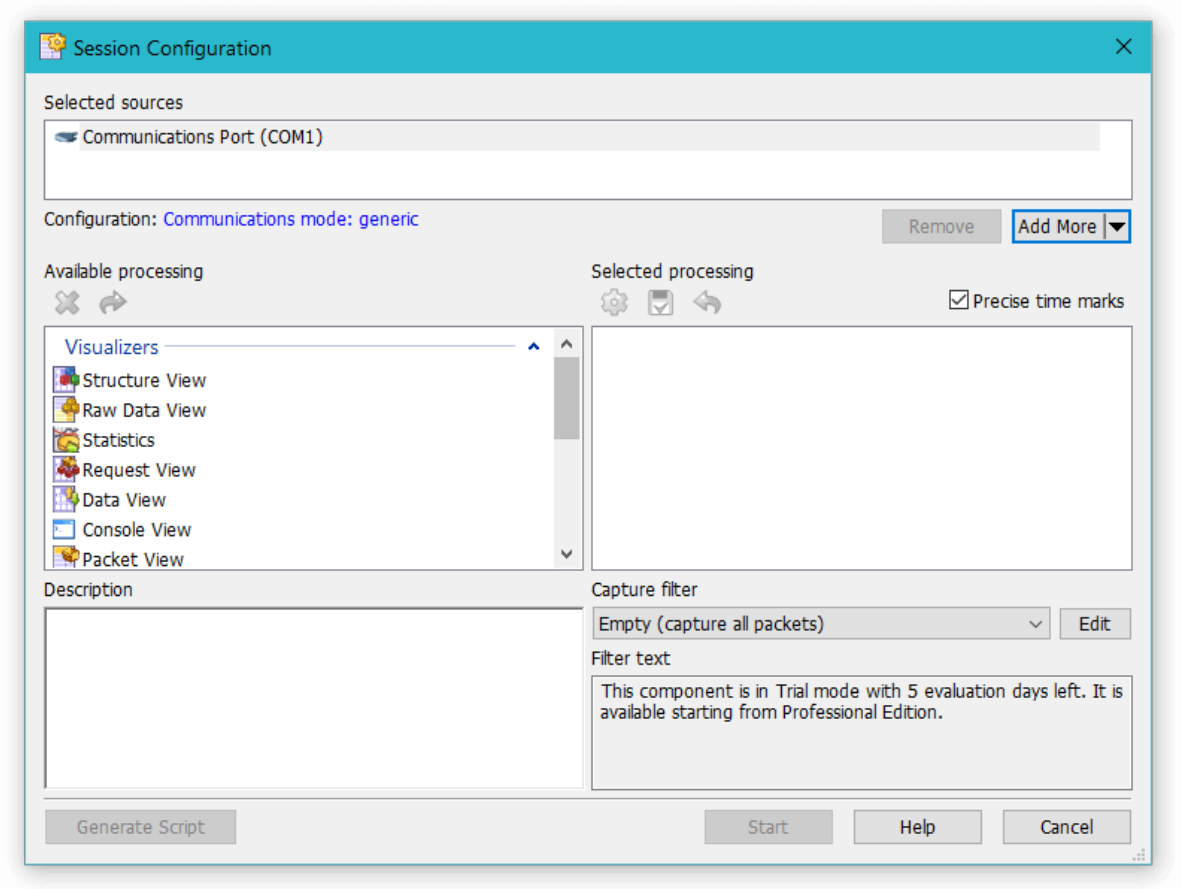
Expands the current session (shows all configured processing modules).

Collapse Item

Collapses the current session (hides all configured processing modules).

Session Configuration Window

This dialog allows you to specify configuration for new monitoring session or change configuration of the existing session.



Selected Sources

When you start configuring a new monitoring session by double-clicking on the device in Devices Tool Window, Session Configuration Window opens. It contains selected device in the **Selected Sources** list.

Device Monitoring Studio supports the so-called multi-source sessions, when two or more devices are included in the same session. The following device types support multi-source sessions:

- Network
- USB
- Serial

The following device types do not support **multi-source** sessions:

- Playback
- Remote (all types)
- Serial Bridge

If the current device type supports **multi-source**, **Add More** and **Remove** buttons appear. Clicking the **Add More** button brings up the list of other devices of the same type. Click on the device to add it to the session. Select the device and click the **Remove** button to remove it from the session.

Running sessions do not allow changing the list of devices. That means that if the Session Configuration Window is opened for a running session, these buttons will not be present.

Configuration

For some sessions, a session-wide device configuration is available. Serial, Playback, Serial Bridge sessions currently provide session-wide configuration. See corresponding sections for more information.

Time Measurement Mode

Device Monitoring Studio supports two time-measurement modes: standard and precise. In standard mode, system default timer is used to measure packet times. Its precision usually varies from 0.5 ms to 20 ms.

Precise mode tells the Device Monitoring Studio to use high-precision system timer, which precision is usually several microseconds. On some computers, using precise timer to monitor a high-baud rate session may decrease system performance. Precise mode is default for Windows 7 or later.

Available Processing

This list contains all installed modules that are available for a current session. Supported modules vary for different device types (USB, Serial or Network) and vary depending on whether the current session is multi-source session or not. Only a small subset of modules support **multi-source** sessions.

Device Monitoring Studio also allows the user to create customized "versions" of predefined data processing modules (see below). If user has created such customized modules, they are also presented in this list.

A data processing module belongs to one of two categories: visualizers and exporters. Visualizers always have a separate designated window to present the result of their work to the user in one or more formats. Exporters usually do not have a designated window and produce non-visible output (like writing to a file). Exporters often communicate with the user by means of Sessions Tool Window.

To add a data processing module to the session, either double-click it in the list, or select and press the **Add to Session** button. A single visualizer may be added multiple times to the session, while some exporters may prohibit this. **Description** pane provides a brief description for the selected processing module.

Favorites

Any processing module may be added to a Favorites list by right-clicking it in the list and selecting the "Add to Favorites" menu option. Favorites are always displayed at the top, allowing for quicker access to the selected processing module.

Selected Processing

Displays a list of data processing modules currently configured for the session.

Some of the modules support additional user configuration. These modules display an underlined comma-separated list of their current configuration settings next to their names. Clicking on this list

brings up the **Configure** window, which allows the user to change default configuration. Alternatively, you may use the **Configure Data Processing...** button.

Note that this configuration is per-module and per-session. That is, if you add multiple Structure View visualizers to a session, you may configure them all with different settings.

Device Monitoring Studio allows you to save the specific configuration for later use. To save custom configuration, follow this procedure:

1. Add required data processing to the session (see above).
2. Click on processing's configuration and change it.
3. Click the **Save Custom Processing...** button and specify the custom configuration name. Custom configuration appears in the **Available Processing** list.

You may now use this custom configuration when you create new monitoring session, for example.

> **NOTE**
>
> Some data processing modules may disallow changing their configuration if session is already running.

To remove data processing module from **Selected Processing** list, select it and press the **Remove** command.

Capture Filter

Allows you to select one of predefined capture filters for a session or create a new capture filter.

Scripting Support

Device Monitoring Studio has the Monitoring Object. It allows user scripts to create, configure and start monitoring sessions and provides full functionality of the **Session Configuration Window** to user scripts.

Pressing the **Generate Script** button will generate a new script file with script code that creates and initializes a monitoring session that matches the current configuration in **Session Configuration Window**.

Device Types

Network

This data source supports monitoring of network traffic through network adapters. It enumerates all installed network adapters, including virtual adapters.

By default, full traffic through the network adapter is always monitored. Use the Capture Filter to reduce the amount of monitored data and filter unneeded traffic.

Currently, the following data processing modules are supported for network monitoring:

- Structure View
- Raw Data View
- Statistics
- Data Recording
- Raw Exporter
- Text Exporter

Multi-Source Support

Network data source as well as all its data processing modules fully support multi-source sessions.

Process Matching

During monitoring, a network source tries to identify a source (for outgoing packets) or recipient (for incoming packets) of the packet. If it succeeds, it stores the process ID and process name. This information is available to the user by means of data visualizers, such as Structure View. This information is organized in such a way, that even if you record a log file and then play it back, correct process association information is displayed to the user. Of course, the process association information is valid **only** on packet capture time and may become invalid any time later. Windows actively reuses process IDs, so care must be taken when the user tries to associate a packet with currently running process.

Protocol Definitions

All network data processing modules such as parsing in Structure View or in Capture Filter depend on protocol definitions installed with Device Monitoring Studio.

These definitions are written using the Protocol Definition Language and are installed with a product in `%INSTALLDIR%\protocols` folder.

Device Monitoring Studio also allows the user to customize these definitions by adding new protocols or modifying existing ones. The current version, although, does not provide any utilities that help the user in this customization. The user needs to edit the supplied protocol definition files manually to achieve his task. This will be addressed in future versions.

USB

This data source supports monitoring of traffic to or from any USB device, connected to a local computer (or remote computer in case of Remote Source).

USB Data Source extracts and presents all available information from a USB device, such asDevice Descriptor, Configuration Descriptor, HID Descriptor and Dependent Devices.

It supports Capture Filter that allows the user to filter unneeded traffic.

Currently, the following data processing modules are supported for USB monitoring:

- Structure View (supports multi-source sessions)
- Raw Data View (supports multi-source sessions)
- URB View (supports multi-source sessions)
- Packet View (supports multi-source sessions)
- Statistics (supports multi-source sessions)
- Custom View (supports multi-source sessions)
- Audio View
- Video View
- HID View
- Mass Storage View
- Still Image View/MTP View
- Communications View
- Data Recording
- Raw Exporter
- Text Exporter

Multi-Source Support

USB data source fully supports multi-source sessions. Not all data processing modules support multi-source session (see the list above).

Protocol Definitions

Structure View, Raw Data View and Custom View data visualizers, as well as USB Capture Filter depend on protocol definitions installed with Device Monitoring Studio. Other data visualizers ("legacy" visualizers ported from the Device Monitoring Studio 6) do not depend on protocol definitions and therefore, do not support user-defined protocols.

These definitions are written using the Protocol Definition Language and are installed with a product in `%INSTALLDIR%\protocols` folder.

Device Monitoring Studio also allows the user to customize these definitions by adding new protocols or modifying existing ones.

Serial

This data source supports monitoring of traffic to or from any serial device, including legacy, PnP and virtual serial devices, connected to a local computer (or remote computer in case of Remote Source).

Serial Source extracts the serial device capabilities and presents them in Serial Device Information window.

It supports Capture Filter that allows the user to filter unneeded traffic.

Currently, the following data processing modules are supported for serial monitoring:

- Structure View (supports multi-source sessions)
- Raw Data View (supports multi-source sessions)
- Request View (supports multi-source sessions)
- Packet View (supports multi-source sessions)
- Statistics (supports multi-source sessions)
- Console View (supports multi-source sessions)
- Custom View (supports multi-source sessions)
- Data View
- MODBUS View
- Line View
- PPP View
- Data Recording

- Raw Exporter
- Text Exporter

Multi-Source Support

Serial data source fully supports multi-source sessions. Not all data processing modules support multi-source session (see the list above).

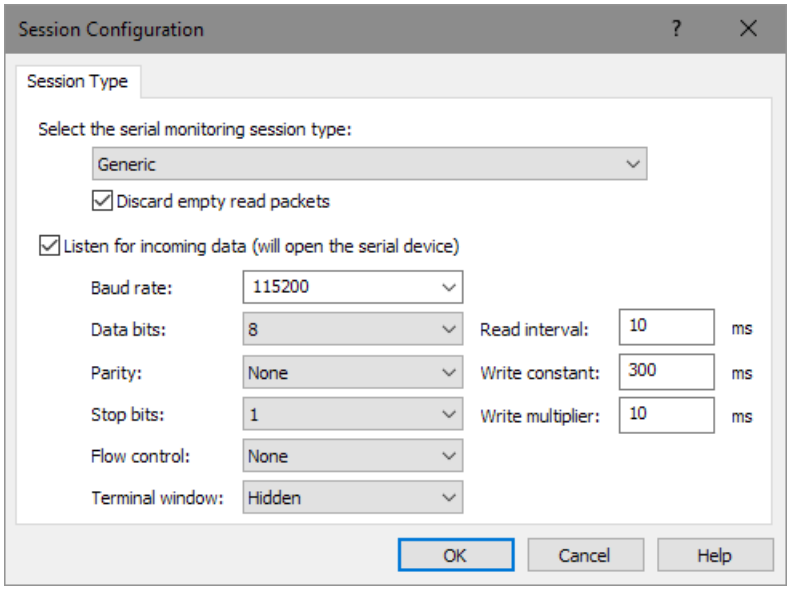
Protocol Definitions

Request View, Console View, Custom View, Structure View and Raw Data View data visualizers, as well as serial Capture Filter depend on protocol definitions installed with Device Monitoring Studio. Other data visualizers ("legacy" visualizers ported from the Device Monitoring Studio 6) do not depend on protocol definitions and therefore, do not support user-defined protocols.

These definitions are written using the Protocol Definition Language and are installed with a product in `%INSTALLDIR%\protocols` folder.

Device Monitoring Studio also allows the user to customize these definitions by adding new protocols or modifying existing ones.

Serial Session Configuration



Serial Source is somewhat different from other data sources. All data sources carefully capture packets originated in monitored application. In case of serial communications, monitored application often does not have a priori knowledge of the protocol and therefore cannot issue write (and especially read) requests of correct sizes. Monitored applications usually read a single packet in several requests or, sometimes, read several packets in a single request. This makes protocol binding and data analysis extremely difficult.

Device Monitoring Studio allows you to specify the so-called serial communication mode, also known as session type. By choosing communication mode, you give DMS a knowledge of the communication protocol and it starts searching for full protocol packets, or frames in the monitored data stream. It then reorganizes the data stream into correct sequence of read and write requests. When the new data stream reaches configured data visualizers and other data processors, they will deal with perfectly formatted frames, and provide accurate information themselves.

Currently, the following communication modes are supported:

Generic

No repackaging occurs. This is a default mode. Additionally, the user may specify if empty read packets should be discarded. This should be selected if monitored application uses active polling: issues a non-blocking read request that gets completed immediately even if there is no incoming data.

PPP

Serial Source searches for PPP (Point to Point Protocol) frames in the monitored data stream. In addition to separating the original data stream into individual PPP frames, it also provides character un-escaping.

One packet in a line

Includes a large set of ASCII protocols where each packet ends with a CR (0D) character, optionally followed by LF (0A) character. This includes such protocols as AT commands (modems), NMEA (GPS devices), MODBUS (ASCII mode) and so on.

MODBUS (RTU mode)

MODBUS RTU mode.

MODBUS (ASCII mode)

MODBUS ASCII mode.

Custom communication mode

This mode allows the user to specify custom rules for packet splitting and joining. Refer to the separate topic Custom Communication Mode for detailed information. If this option is grayed out, make sure Scripting feature is installed. PPP frames usually encapsulate a network protocols inside them. Using PPP communication mode and PPP View data visualizer, allows you to automatically decode them.

Listening Mode

Serial source integrates with Serial Terminal. You may have the serial monitoring session automatically start a terminal session for the selected device. Enable the "Listen for incoming data" option and configure the port parameters, such as baud rate, data bits, parity, stop bits and flow control.

If you intend to communicate with a device from the Device Monitoring Studio, make sure the "Terminal window" switch is set to "Visible".

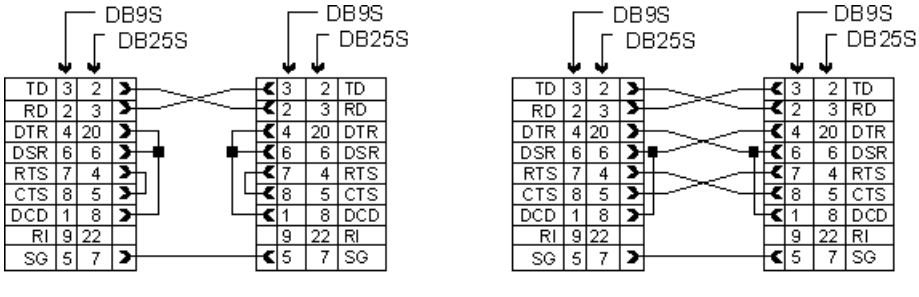
Serial Bridge

This module allows the user to create virtual bridges between two serial devices.

For example, if a bridge is created between COM1 and COM2 and then two devices are connected to COM1 and COM2 correspondingly, it appears to them as if they are connected directly to each other. At the same time, the presence of virtual bridge allows Device Monitoring Studio to capture the traffic exchanged by those devices.

Devices must be connected to physical ports using the so-called null-modem cables. The actual cable wiring depends on device type and configuration. In some cases, two null-modem cables are required while in others two straight or one null-modem and one straight cables are required.

Null-modem cable is a simple cable with two jacks, which connects two RS-232 ports of two DTEs. It connects grounds, RD and TD cross-wise, and other signals are connected according to one of the schemes provided below. The set of transferred channels can be reduced, depending on the flow control being used, in most cases you need as much as only three wires (ground, RD and TD):

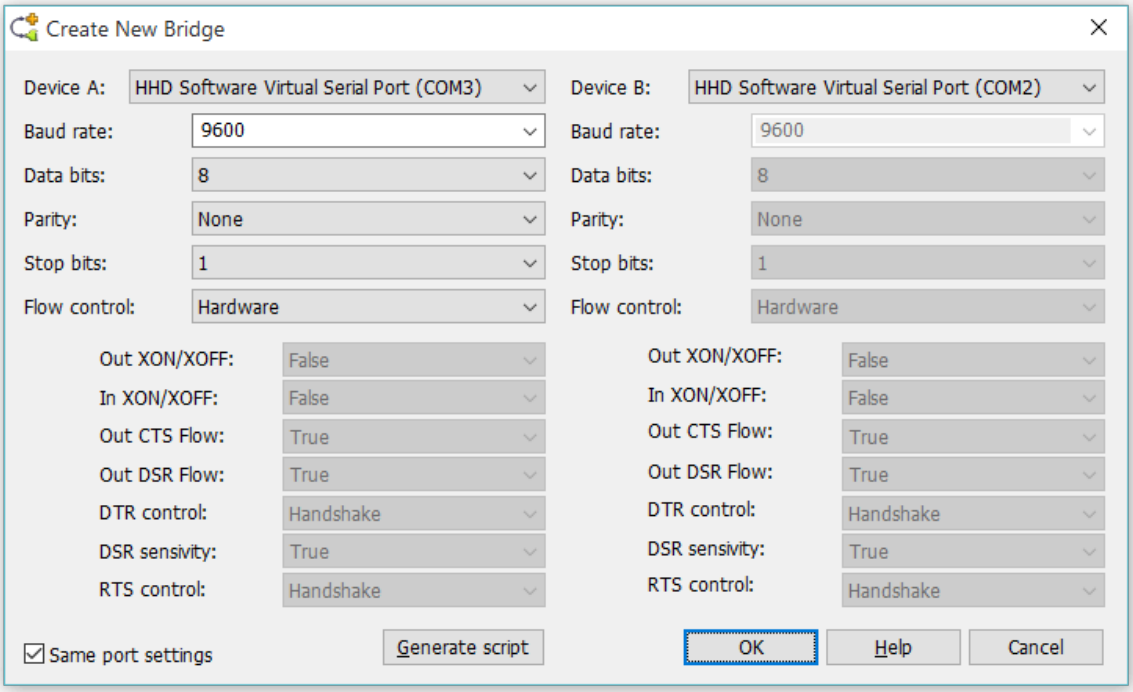


Null-modem cable: a - minimum, b - full

A bridge is active only when a monitoring session is running for the bridge.

To create a bridge, either press the **Create New Bridge...** command on the Devices Tool Window's toolbar or execute the **Bridge » Create New Bridge...** command.

You need at least two serial devices to create a bridge. The following window appears after you execute the command:



Specify the settings for a new bridge and click the **OK** button. A window appears that asks for a bridge name. Enter the bridge name and click the **OK** button.

Once created, a bridge is always present in Devices Tool Window under the “Bridges” subcategory of “Serial” category. You may rename a bridge, change its configuration or remove a bridge using the commands in the Bridge main menu or in bridge's context menu.

In addition, when you start a new monitoring session, Device Monitoring Studio allows you to “temporary” change the bridge configuration only for the duration of this session. “Default” bridge configuration is not affected in this case.

Serial Bridge does not support Capture Filter.

Currently, the following data processing modules are supported for bridges:

- Raw Data View

- Request View
- Packet View
- Statistics
- Data View
- PPP View
- Data Recording

Generating Script

Click the Generate script button to generate script that creates a new serial bridge and sets its parameters according to settings in the window.

Multi-Source Support

Serial Bridge does not support multi-source sessions.

Timeout Configuration

TODO

Communications Mode

Serial Bridge Source is somewhat different from other data sources. All data sources carefully capture packets originated in monitored application. In case of serial bridge, Device Monitoring Studio creates a virtual link between two serial devices and forwards data between them. It does not have a priori knowledge of the protocol and therefore cannot issue read and write requests of correct sizes. Device Monitoring Studio uses pre-allocated buffers and default timeout settings and often reads several packets in a single request, or, sometimes reads a single packet in several requests or. This makes protocol binding and data analysis extremely difficult. Moreover, by default, Serial Bridge does not support protocol binding at all.

Device Monitoring Studio allows you to specify the so-called *serial communication mode*. By choosing communication mode, you give DMS a knowledge of the communication protocol and it starts searching for full protocol packets, or frames in the monitored data stream. It then reorganazies the data stream into correct sequence of read and write requests. When the new data stream reaches configured data visualizers and other data processors, they will deal with perfectly formatted frames, and provide accurate information themselves.

Currently, the following communication modes are supported:

Generic

No repackaging occurs. This is a default mode. Protocol binding is not available in this mode. Raw Exporter and Text Exporter visualizers are not available in this mode too.

PPP

Serial Bridge Source searches for PPP (Point to Point Protocol) frames in the monitored data stream. In addition to separating the original data stream into individual PPP frames, it also provides character un-escaping.

One packet in a line

Includes a large set of ASCII protocols where each packet ends with a CR (0D) character, optionally followed by LF (0A) character. This includes such protocols as AT commands (modems), NMEA (GPS devices), MODBUS (ASCII mode) and so on.

MODBUS (RTU mode)

MODBUS RTU mode.

MODBUS (ASCII mode)

MODBUS ASCII mode.

PPP frames usually encapsulate a network protocols inside them. Using PPP communication mode and PPP View data visualizer, allows you to automatically decode them.

Playback

Playback Data Source allows you to play back the log files recorded using the Data Recording processing module. Playback never sends any data to any physical device; it only "mimics" the original device and allows the user to replicate the original monitored session. Log file playback is useful if session analysis is hard or impossible in real-time during actual device monitoring.

Device Monitoring Studio has two globally configured paths: *path to log files* and *default path for playback*. By default, these paths both point to the same folder and this is a recommended configuration. The user may change these paths on the Tools » Settings, Recording/Playback Tab.

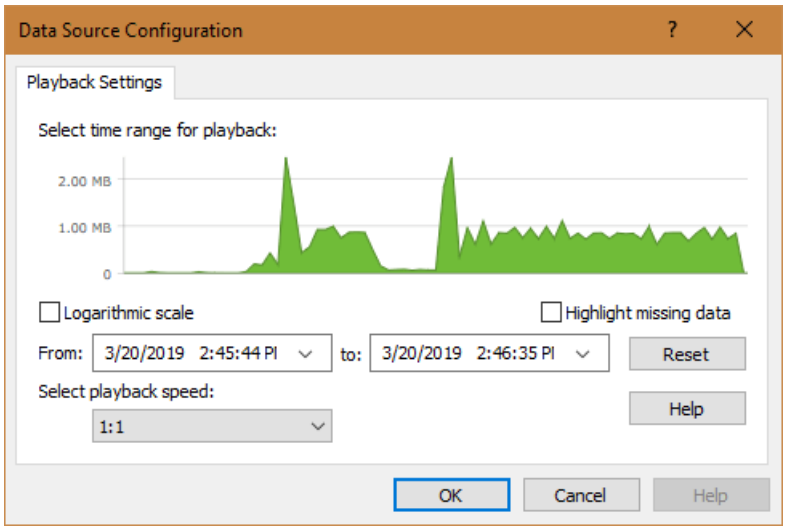
Data Recording uses the log folder to store log files it creates. **Playback Data Source** scans the playback folder for log files and adds them to corresponding categories in Devices Tool Window. A Recorded Sessions category is created as a child of USB, Serial or Network categories, into which log files are added. For convenience, device log files are placed into the category named after the device.

Device Monitoring Studio automatically watches for changes in the *playback folder* and updates the tree accordingly.

Starting Playback

To start log file playback, select a log session in a tree and press the **Start Monitoring...** button or just double-click it.

A **Data Source Configuration** window opens that allows you to select the time range for playback as well as specify the playback speed.



Use the *From* and *To* time selection fields to specify the required range, or use the mouse to select the range on the plot.

You may select playback speeds from 1:16 to 16:1, as well as two additional options: stepped and continuous. These addition options mean the following:

Stepped

Display only a single packet after which requires you to press the Next button on Device Monitoring Studio's status bar.

Continuous

Do not take original time delays between packets into account and displays the whole contents of the log file stream at once. Use this mode with care, as it may lock Device Monitoring Studio's user interface for some time.

Playback Controls

During playback, Device Monitoring Studio's status bar displays two progress bars and optional **Next** button. The first, longer progress bar displays the overall playback progress. If there is a significant delay between two consecutive packets, a second progress bar visualizes this delay. If you do not want to wait, click the **Next** button. In *stepped* playback speed mode, you must press the **Next** button after each packet.

Managing Log Files

You may use additional commands, available through the item context menu in Devices Tool Window to manage log files:

Open File Location

Open the log file's folder in Windows Explorer.

Delete

Delete the log file (AKA session) or all log files for the device.

Sort by Date

Sort all log file sessions by date of creation.

Sort by Name

Sort all log file sessions by their names.

Display Date and Time

Display session's date and time next to its name.

Display Size

Display session's size, in bytes, next to its name.

Working with Log Files from Other Locations

Although Device Monitoring Studio automatically detects and displays log files in default *playback folder*, you may easily play back a log file located in any other location. Just use the **File » Open** command, Drag&Drop or simply double-click the log file in Windows Explorer to open the file in Device Monitoring Studio and bring up the Session Configuration Window.

Multi-Source

Multi-source is not a separate data source type. Instead, it is a technology that allows the user to monitor two or more devices of the same type in a single monitoring session. Device Monitoring Studio considers such monitoring session as having a single source and sorts incoming packets according to their capture times.

The following data sources support multiple-source:

- Network
- USB
- Serial

The following data sources do not support multiple-source:

- Serial Bridge
- Playback
- Remote

Multi-Source Session Creation

You start creating multi-source session just like any other session. Double-click a first device you want to monitor in the Devices Tool Window to bring up the Session Configuration Window.

If the selected device belongs to one of the supported data types, **Add More** button appears. Click the button to see the list of all devices of the same type you may add to the session. You cannot add the same device twice and you cannot exceed the maximum of 16 devices per session.

Not all data processing modules support multi-source sessions. Session Configuration Window automatically updates list of available processing modules when you add or remove device. If only one device remains in a session, it becomes an ordinary session.

After creation, multi-source session does not allow you to add or remove devices.

Multi-Source Device Identification

To help you distinguish packets from different devices, all data visualizers that support multi-source sessions use different colors to highlight packets from different devices. You configure a global color table on Tools » Settings, Multi-Source Device Colors Tab. For reference, Sessions Tool Window uses the same colors when it displays the list of devices in a session.

Unsupported Data Sources

Serial Bridge, Playback and Remote data sources do not support multi-source.

Note that Data Recording processing module supports multi-source, but when you later play back the resulting log file, it is considered as having a single source, but will still show you packets from all original devices.

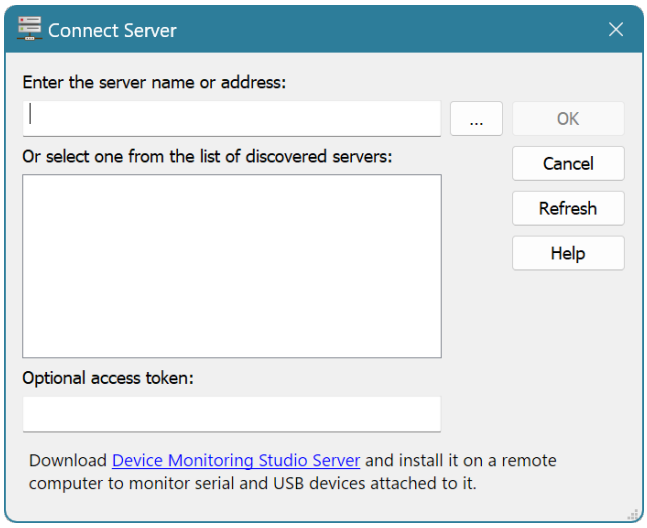
Remote

Device Monitoring Studio supports monitoring of Serial and USB devices, connected to remote server. Device Monitoring Studio with **Remote Data Source** installed already supports monitoring of remote devices, but you need to install a copy of Device Monitoring Studio Server on remote computer to make its devices available for clients.

Remote Source is not a separate data source; instead, it is a technology that allows the user to access Serial and USB data sources running on a remote server.

Connecting to Remote Server

Execute the **Tools » Connect New Server...** or press the **Connect New Server...** button on Devices Tool Window's toolbar. The following window opens:



Enter the server name or address manually or click the **Browse** button to search for the server name in the Active Directory or Local Network or select one of the auto-discovered servers.

Auto-discovery works in domain and workgroup networks and automatically locates all available servers. Note that server administrator can disable server auto-discovery.

After you click the **OK** button, Device Monitoring Studio connects to remote server, queries the list of supported devices and adds them to the **Devices Tool Window**.

Remote Monitoring Session

You start a remote monitoring session just like a local monitoring session. First, select a remote device in Devices Tool Window and press the **Start Monitoring...** button on toolbar, or simply double-click on the device. Session Configuration Window opens. Configure remote session the same way as you configure a local session.

Remote sessions do not support multi-source.

Disconnecting from the Server

To disconnect a server, select its name in Devices Tool Window, right-click to bring up a context menu and select the **Disconnect** item. All server devices are removed from a tree as well as all running monitoring sessions are terminated.

Network-related Errors

Whenever a network error, connection error, server software error or hardware error occurs, a connection is terminated. You receive a corresponding error message. All running monitoring sessions from this server are also terminated with error message.

A client and server exchange a short keep-alive signal every 30 seconds even if there are no running monitoring sessions.

Server Configuration

Consult the corresponding topic to find more information on installing and configuring Device Monitoring Studio Server.

Import

Import Data Source allows playing back network session logs recorded by other applications. The

following applications are supported:

- Wireshark (`.pcap` log files)
- Microsoft Network Monitor (`.cap` log files)

To start an import data source monitoring session, double click the "External log file import adapter" item under "Network » Virtual Adapters" item in Devices Tool Window.

Then click the "Select file" link under the **Configuration** to open up **Data Source Configuration** window where you can specify the log file to use.

Data Processing

Custom View

Device Monitoring Studio supports parsing monitored packets according to a defined protocol (see protocol binding topic for details). After the packet is bound, values of packet fields are available for processing. The easiest way to see all field values is to use Structure View data visualizer.

However, if you want a better format for field values, use the **Custom View** data visualizer. It takes a special script file, written in TypeScript (a superset of JavaScript) which must be placed into the default protocols folder and its file name must end with `.view.ts`.

Custom View Workflow

When Custom View initializes, it loads a user-supplied script file and executes it. By convention, the user script should do nothing in a global scope and should not use global variables. Only declarations of different kinds are allowed at a global scope.

After the script execution ends, a following global function is invoked:

```
TypeScript
function createVisualizer(multiSession: boolean): ICustomVisualizer;
```

It is passed a single `boolean` argument which indicates whether this is a multi-source session or not.

The user script must initialize an instance of a class that implements the `ICustomVisualizer` interface and return a reference to it:

```
TypeScript
interface ICustomVisualizer {

    // Custom visualizer's name
    readonly name: string;

    // Custom visualizer's description
    readonly description: string;

    // A source type (a string) or types (string array).
    // Supported types are "serial", "usb", "bridge" and "network"
    readonly sourceType: string | string[];

    // Optional error scheme ordinal. If omitted, defaults to zero
    readonly errorScheme?: number;

    // Return schemes used by this custom view
    getSchemes(): SchemeDefinition[];

    // Return a list of options supported by this custom view
    getOptions(): OptionDefinition[];

    // processPacket is called with each packet
    processPacket(packet: Object, isJoinPackets: boolean): void;
}
```

All interface members are mandatory and custom class must implement all of them. `name` should be set to unique custom visualizer name, which is later displayed in Protocols List Window. `description` is displayed to the user when he selects a custom visualizer from a list.

`sourceType` must be assigned a string that indicates the supported session type, that is, “serial”, “usb”, “bridge” or “network”. If the visualizer supports several session types, the property must be assigned to an array of strings indicating all supported session types.

`getSchemes` method is invoked to retrieve an array with all defined visual schemes. `SchemeDefinition` is

defined as

```TypeScript
interface SchemeDefinition {
    // Scheme name
    name: string;
    // Font face
    fontFace: string;
    // Font size, in pt
    fontSize: number;
    // Font weight is bold
    bold?: boolean;
    // Font style is italic
    italic?: boolean;
    // Text color
    color?: number;
    // Background color
    backColor?: number;
}
```

All fields are self-explanatory. Note that font size is in *points* and may be a non-integer number. Colors must be represented as a hexadecimal number in a form of `0xRRGGBB`. Device Monitoring Studio internally stores the list of schemes and provides the user with an interface to change colors and fonts. Subsequently, custom script refers to the individual visual scheme by its ordinal number in an array returned by the `getSchemes` method. The array the method returns is required to contain at least one visual scheme.

Optional `errorScheme` property may specify the scheme ordinal to be used to display system error messages. If omitted, defaults to zero.

`getOptions` method should return (a possibly empty) list of user-controlled options for a custom view script. `OptionDefinition` is defines as

```TypeScript
interface OptionDefinition {
    // Option friendly name
    name: string;
    // Default option value
    value: boolean;
    // Change function, called with a new value when user changes option
    change: (newVal: boolean) => void;
}
```

As with visual schemes, the list of options is stored internally and the user is provided with a way to change option values. Option values are allowed to be changed at any time and they are automatically applied with a call to the functor passed in a `change` property.

`processPacket` method is then invoked for each bound packet. A reference to a packet is passed in a first method argument, while the current state of a global "Join consequent packets" switch is passed in a second argument.

WARNING

The custom script should not make any assumptions on the order of incoming packets. For performance reasons, the consequent order of packets is not guaranteed. Therefore, custom view script cannot store any inter-packet state inside a class that derives from `ICustomVisualizer` interface.

The following topic, the Visualizer Host describes the API available to custom view script to process and format packet data.

Visualizer Host

Accessing Fields of a Bound Packet

```TypeScript
function get(obj: object, path: string, forceReference?: boolean): any;
```

A first argument to a `processPacket` method is a reference to an object that represents a bound packet to a custom view script. For performance reasons, parsed fields are not directly accessible to the script. Instead, it must call a `get` function, passing it the bound packet object, a full path to the required field and an optional `boolean` argument, which, being set to `true`, forces the evaluation of the ref() function for the result.

Advanced Formatting

```TypeScript
function format(fmtString: string, ...values: any[]) : string;
```

`format` function takes a format string and a list of arguments to substitute in a format string and returns a resulting string.

Visualizer Host

Visualizer Host is an object accessible to a custom view script via a global variable `nvis`. It implements the following interface:

```
TypeScript
interface IVisualizerHost {
    // Enable or disable line numbers and optionally set scheme for line numbers
    enableLineNumbers(state: boolean, scheme?: number): void;

    // Add new block, optionally set background color
    addBlock(color?: number): void;

    // Add text. '\n' character is used to separate lines
    addText(text: string, scheme?: number): void;

    // Low-level add text
    addTextRaw(...args: (string | number | boolean)[]): void;

    // Add a line break
    addNewLine(): void;

    // Add a dump
    addDump(scheme: number, obj: object): void;

    // Add a formatted table
    addTable(nameScheme: number, valueScheme: number, ...rows: any[]): void;

    // Format packet time
    formatTime(entryTime: number, prevEntryTime: number): string;

    // Get multi-source device name
    getDeviceName(ordinal: number): string;

    // Visualize bound field
    visualize(obj: object): string;


    // Control flow: set variable
    setVariable(varName: string, varValue: number): void;
    // Control flow: if
    ifEq(varName: string, varValue: number): void;

    // Control flow: if not
    ifNotEq(varName: string, varValue: number): void;

    // Control flow: end if
    ifEnd(): void;

    // Control flow: else
    ifElse(): void;
}
```

`enableLineNumbers`

This method switches the built-in displaying of line numbers for a custom view. This is the only method of the `IVisualizerHost` interface that is allowed to be called outside of the execution of `processPacket` method.

`addBlock`

Add a new block to the visualizer. A block is a collection of lines. Block may have a separate background color, which may be directly passed to this method. If omitted, a color is set automatically. By default, the Custom View uses interlacing when displaying subsequent blocks.

`addText`

Add a text to the visualizer. A text may contain one or more lines, separated by the `'\n'` character. A second optional argument specifies the visual scheme (as an ordinal in visual scheme array) to be used to display the text. If omitted, the current scheme is used.

`addTextRaw`

A more efficient version of the previous method. It accepts any number of arguments. Each argument must be a `string`, a `number` or a `boolean`. If an argument is a string, it is added to the visualizer. Note that the string is not scanned for new line characters. A number changes the current

visual scheme to a given one. A special value of -1 reverts the most recent visual scheme change. A boolean value (no matter whether it is true or false) adds a newline.

`addNewLine`

Add a new line to the visualizer.

`addDump`

Add a dump to visualizer. The first argument indicates a visual scheme to use and the second argument must be a reference to a bound packet field you want to display. It is recommended to use monospace font, otherwise the dump will look unaligned.

`addTable`

Add a formatted table to the visualizer. The first argument is a visual scheme index to use for a field name and the second argument is a visual scheme to use for a field value. Other arguments must come in pairs. The first element of a pair is a field name (a string), while the second element is a field value. If you want to use the special visualization algorithms (also used by Structure View and Text Exporter components), like automatic parsing of enumeration values and automatic array visualization, make sure you pass a reference to a field. For simple cases, use plain field values.

`formatTime`

Standard packet time formatting routine. Pass it the current packet's time and previous packet's time to get a formatted packet time string.

`getDeviceName`

May be used in multi-source session to obtain a name of a source device by its index.

`visualize`

Invokes standard visualization mechanism for a passed field reference and returns a resulting string.

`setVariable`

Assigns a value for a given variable. As mentioned before, the order of packets in calls to `processPacket` is not guaranteed, preventing the custom view script of maintaining any inter-packet state. To overcome this limitation, a custom view script may have any number of named variables, access to which is synchronized. That means, that variable set during processing of packet `N` is guaranteed to have the same value during processing of packet `N+1`.

`ifEq`

Checks if the variable equals the given value. If the condition is false, all subsequent visualizer commands are ignored until `ifElse` or `ifEnd` method is invoked.

`ifNotEq`

Checks if the variable not equals the given value. If the condition is false, all subsequent visualizer commands are ignored until `ifElse` or `ifEnd` method is invoked.

`ifEnd`

Marks the end of a current if block.

`ifElse`

Reverts the result of the previous condition for subsequent commands, until the `ifEnd` method is invoked.

Samples

For an up-to-date samples, look for source code for Request View, Console View in Serial Monitor and HTTP View in Network Monitor.

The Protocols List Window may be used for fast navigation to the source code.

User Experience

Custom View data visualizer window supports all basic navigation operations: it may be scrolled using mouse, keyboard or with help of scrollbars. It also supports scroll touch gestures.

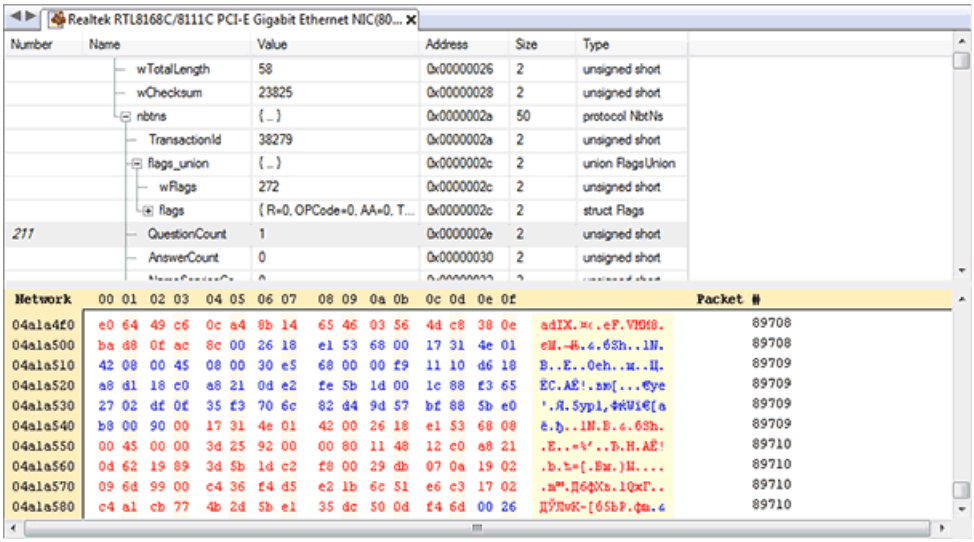
Custom View supports selection, where a portion of text in a window (or the whole window contents) may be selected using mouse. Double-clicking a word selects the word. Selected text may later be copied into the Clipboard or exported into an external file in text or HTML format.

The **Edit » Find** command may be used to search for a pattern in a window. **Edit » Go to Packet** command jumps to the given packet (or the closest available) and highlights it.

Edit » Select Packet selects the entire packet under the mouse cursor.

Structure View

Structure View data visualizer is a two-part window. First part displays each monitored packet parsed according to installed set of protocols. Second part displays raw packet contents. Changing the current position in one window automatically changes the position in another, provided the **Tools » Raw Data View » Synchronize** option is turned on.



Decoded Packet Contents

Each packet is matched against one or more of loaded protocol definition files and if matched successfully, all protocol fields are shown in the top part of the Structure View visualizer. For each field, its name, value, starting offset and size are displayed. If field consists of other fields, you may expand it by pressing the little plus icon or using the `Right Arrow` key on the keyboard.

You may use mouse and keyboard to navigate to other fields within a current packet or to other packets.

If you bring up the context menu, the following commands are displayed:

Copy Line

Copies the currently selected line into the Clipboard.

Copy Value

Copy only field's value into the Clipboard.

Expand Sub-Tree

Expand the field's sub-tree.

Collapse Sub-Tree

Collapse the field's sub-tree.

Next Packet

Jump to the next packet.

Previous Packet

Jump to the previous packet.

Go to Packet...

Jump to the specified packet.

Selected Packet Details...

Display details for the selected packet (if Synchronize option is off).

Set Filter...

Set or modify the current display filter.

Raw Data View

See the Raw Data View visualizer section for description of the bottom part of the Structure View data visualizer.

Root Protocol

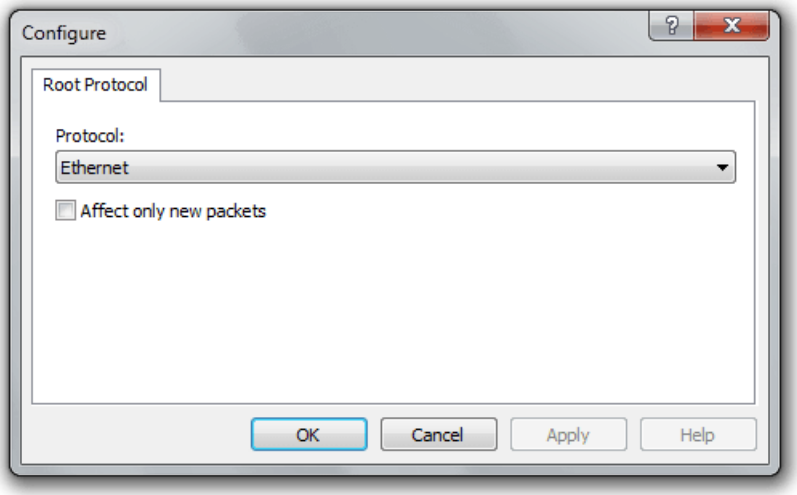
By default, Structure View visualizer always uses the predefined "root" protocol when it parses incoming packets. The following table describes default root protocols for each supported source type:

Type	Default Protocol Name
Network	`Ethernet`
USB	`Usb`
Serial	`Serial`

Each individual Structure View window allows you to select another protocol to become a root for each packet. If monitored packet does not belong to the selected protocol, it is never displayed, thus the root protocol may also behave like a filter.

To change the root protocol, do one of the following:

- Before session creation: after you add a **Structure View** processing module in the Session Configuration Window, open its configuration and switch to the **Root Protocol** tab.
- For running session: click on **Structure View**'s configuration settings in Sessions Tool Window and switch to the **Root Protocol** tab.
- For running session: right-click in the visualizer to bring up the context menu and select the **Set Root Protocol...** item.



Select the protocol from a list. If you are changing the setting for a running session, use the **Affect only new packets** switch to control the change behavior.

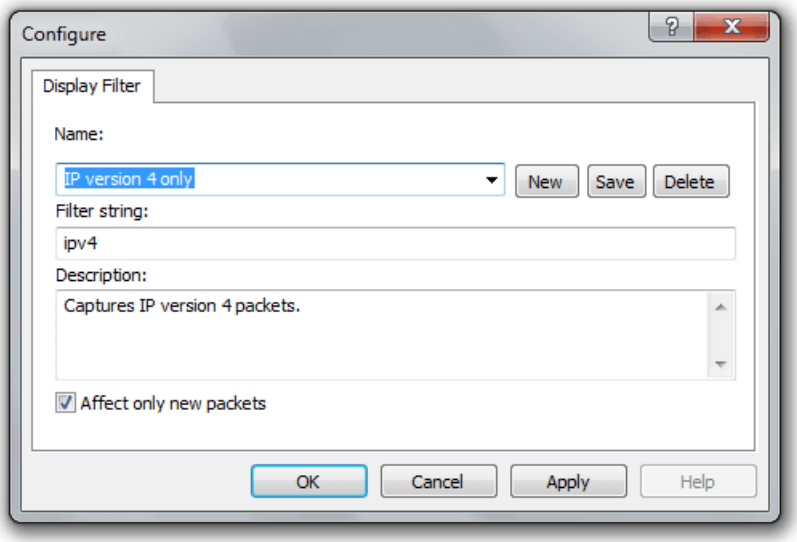
If you leave the switch unchecked, the session will be paused until the Structure View visualizer finishes updating.

Display Filter

In addition to session-wide Capture Filter, each individual Structure View window allows you to set its own filter, called *display filter*.

To change the display filter, do one of the following:

- Before session creation: after you add a Structure View processing module in the Session Configuration Window, open its configuration and switch to the **Display Filter** tab.
- For running session: click on **Structure View**'s configuration settings in Sessions Tool Window and switch to the **Display Filter** tab.
- For running session: right-click in the visualizer to bring up the context menu and select the **Set Filter...** item.



Use this window to select a display filter from a list, or enter the filter string manually, give it a name and save.

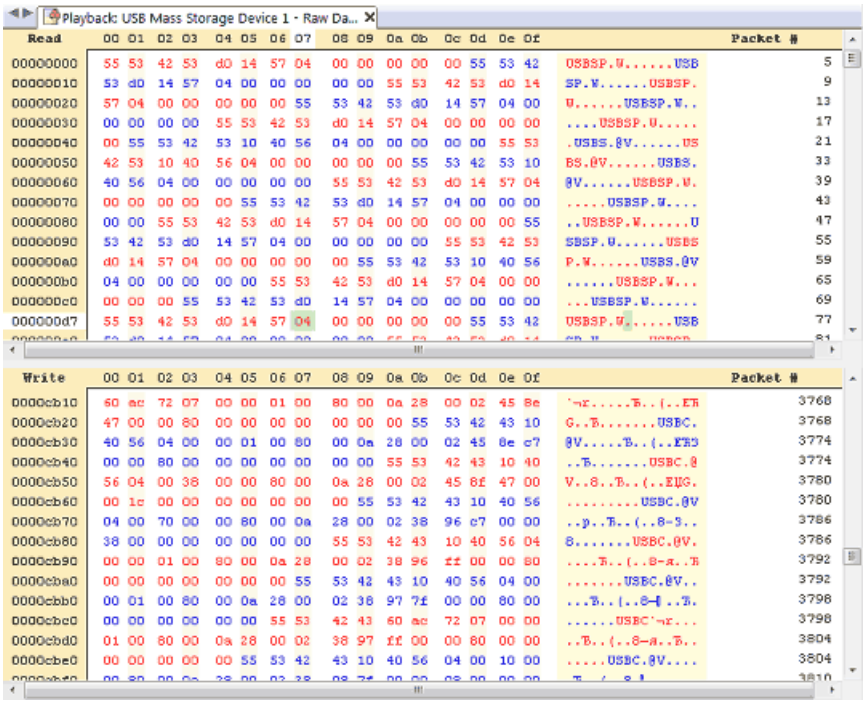
If you are changing the setting for a running session, use the **Affect only new packets** switch to control the change behavior. If you leave the switch unchecked, the session will be paused until the Structure View visualizer finishes updating.

Operation

Structure View data visualizer with configured **Display Filter** parses each incoming packet according to current protocol definition files and then evaluates a filter string against it. If a packet passes the test, it is included in the visualizer, otherwise, it is discarded.

Raw Data View

The purpose of this data visualizer is to extract the raw data from packets and display it as contiguous data stream.



The visualizer is visually divided into two parts, where incoming data is displayed in the top part and outgoing data is displayed in the bottom part. You can use the mouse to adjust the size of both parts.

All incoming or outgoing data is grouped into the single stream. By default, each next packet has interlaced background color, allowing you to visually separate packets from one another. You can choose between three packet coloring modes using the **Tools » Raw Data View » Packet Coloring** command. There are “no coloring” mode, “odd/even” coloring mode and “read/write” coloring mode.

When you hover the mouse pointer over data in a stream, the following brief packet information is displayed:

1. Packet number.
2. Packet absolute time and difference from the previous packet.

> **NOTE**
>
> Time difference to the actual previous packet is displayed. This maybe an informational or control packet, or packet, which is actually displayed in another part (that is, outgoing or incoming packet).

3. Size of the whole packet.
4. Size of the data taken from the packet.
5. For USB, the endpoint information is displayed.

The cursor movement in both parts may be synchronized. After you enable the synchronization using the **Tools » Raw Data View » Synchronize** command, moving the cursor in one part automatically moves the cursor in another part. Device Monitoring Studio automatically locates the nearest packet.

Customization

The **Raw Data View** provides great customization options, which you may assign separately to incoming and outgoing parts.

Display As

You may configure data display as hexadecimal, decimal, octal or binary numbers. You can also treat data stream as floating-point numbers of either single or double precision.

Group By

You may group data in bytes, words, double words or quad words.

Columns

You may change the number of displayed columns. If you select “Auto”, the data stream will take all available window width.

Code Pane

Allows you to switch the code (left) pane on or off.

Text Pane

Allows you to switch the text (right) pane on or off.

Endianness

You may display data as little-endian or big-endian. Obviously, this will not affect data grouped in bytes.

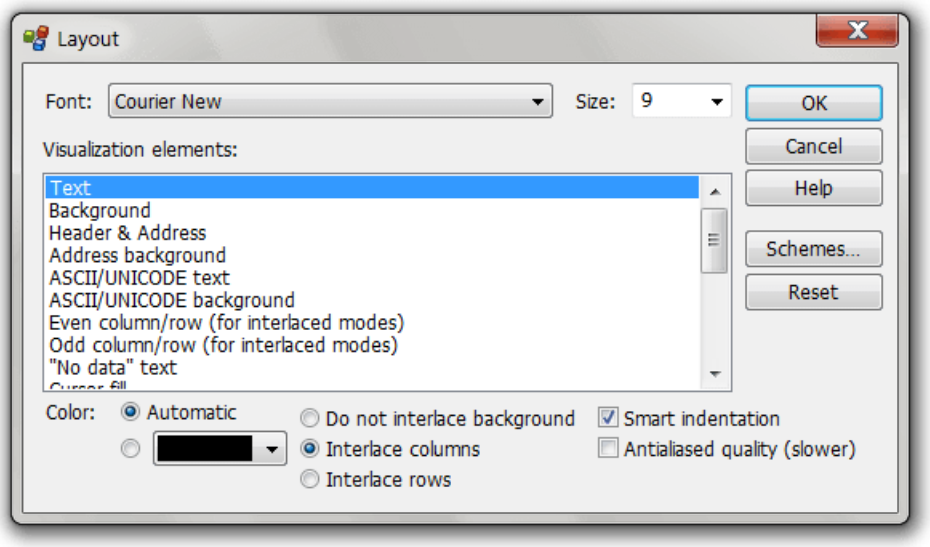
Encoding

Change the text encoding in the text pane. More than 160 different encodings are supported, including UTF-8 and UTF-16.

Endpoints (USB only)

Allows you to filter the data stream by USB endpoints.

In addition, the **Raw Data View** visualizer allows you to fine-tune the visual aspect of many features it displays. Bring up the context menu and select the **Coloring...** command. The following window appears.



The following visualization elements are accessible for customization:

Text

Foreground color of all text.

Background

Background color of all text.

Address

Foreground color of address area.

Address background

Background color of address area.

ASCII/UNICODE text

Foreground color of text pane.

ASCII/UNICODE background

Background color of text pane.

Even column/row background (for interlaced modes)

Background color for column/row interlaced mode

Odd column/row background (for interlaced modes)

Background color for column/row interlaced mode

“No data” text

Foreground color of the area after the last monitored byte.

Cursor fill

Background color of the cursor.

Cursor outline

Cursor's outline color.

Active row & column highlight text

Foreground color of a highlighted cell.

Active row & column highlight background

Background color of a highlighted cell.

Odd packet text

Foreground color of an odd packet if packet interlacing mode is on.

Odd packet background

Background color of an odd packet if packet interlacing mode is on.

Odd packet outline

Outline color of an odd packet if packet interlacing mode is on.

Even packet text

Foreground color of an even packet if packet interlacing mode is on.

Even packet background

Background color of an even packet if packet interlacing mode is on.

Even packet outline

Outline color of an even packet if packet interlacing mode is on.

Selection

Background color of a selection.

Some of these elements allow you to select “Automatic” option. In this case, they will either use the color from another related element, or use operating system default color.

The following options are also configurable through the Layout Dialog:

Packet interlacing

You may choose to interlace rows, colors or do not interlace at all.

Smart indentation

More visually appealing indentation is applied in some group modes when this setting is on.

Antialiased quality

Use Cleartype fonts to display visualizer data. If you enable this option, Device Monitoring Studio may require more processing resources.

You may save current settings as schemes to load them in the future. HHD Software also provides several schemes for you to choose.

Navigation

You may use standard keyboard keys to navigate the view:

Key or Key Combination	Action
`Left`	Move the cursor one cell left. If the cursor is on the first cell in a line, it goes to the last cell on the previous line.
`Right`	Move the cursor one cell right. If the cursor is on the last cell in a line, it goes to the first cell on the next line.
`Up`	Move the cursor one line up.
`Down`	Move the cursor one line down.
`Home`	Move the cursor to the beginning of the line.
`End`	Move the cursor to the end of the line.
`Ctrl + Home`	Move the cursor to the beginning of the data stream.
`Ctrl + End`	Move the cursor to the end of the data stream.
`PgDn`	Scroll the data stream one page down.
`PgUp`	Scroll the data stream one page up.
`Ctrl + Right`	Go to the beginning of the next packet (default combination)
`Ctrl + Left`	Go to the beginning of the previous packet (default combination)

In addition, you may use the mouse to navigate the cursor or scrollbars to navigate a view.

The following command, in addition to keyboard and mouse navigation, may be used to navigate the view:

Tools » Raw Data View » Go to Offset

Navigate to the offset in the window. Offset may be specified as a decimal or hexadecimal number or as percentage.

Tools » Raw Data View » Previous Packet

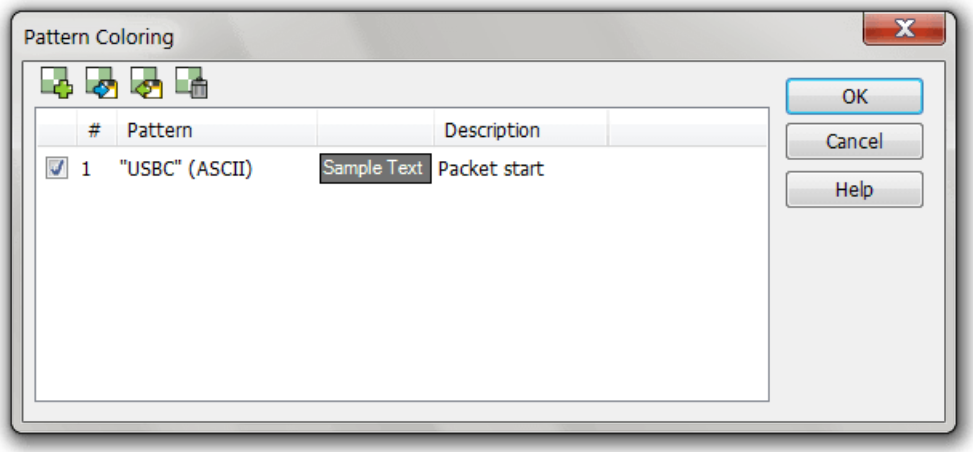
Navigate to the beginning of the previous packet.

Tools » Raw Data View » Next Packet

Navigate to the beginning of the next packet.

Pattern Coloring

Raw Data View visualizer allows you to create one or more patterns to highlight them in the data stream automatically. A pattern is either an encoded or UNICODE string, regular expression or a sequence of bytes, words, double words or quad words.



For each pattern you create, you may select the background and foreground color as well as an optional outline color. You may also specify a transparency value for a background color so the colors from multiple components mix well. A single cell may have the following applied coloring:

1. If packet interlacing option is on, odd and even packets are colored differently.
2. If there is a selection, cells belonging to a selection are colored differently.

This window also allows you to save the rules to a file or load them from a file.

See the Regular Expressions topic for details on supported regular expression syntax.

Advanced

Selecting Data

The view supports the concept of multiple selection, first introduced in our Hex Editor Neo product. That means that several non-contiguous ranges of data may be selected at the same time. We refer you to the documentation of the Hex Editor Neo for more information about the multiple selection concept and how to work with it.

You may use both mouse and keyboard to select data. Hold the **Shift** key while navigating to modify the current selection. Hold the left mouse button and drag the mouse to do the same. Hold the **Ctrl** or **Alt** keys to change mouse selection behavior.

To select the whole packet the cursor stays in, execute the **Edit » Select Packet** command.

You can select all data in a stream using the **Edit » Select All** command and drop the selection using the **Edit » Select None** command. You can invert the current selection using the **Edit » Select Invert** command.

Another way to select data is to execute the **Find All** command. See Searching for Data for more information.

Exporting Data

After you selected data in a view, you may copy it to the Clipboard or export to the file.

Copying to the Clipboard

After you execute the **Edit » Copy** command, depending on whether the cursor is currently in Code Pane or Text Pane, the following occurs:

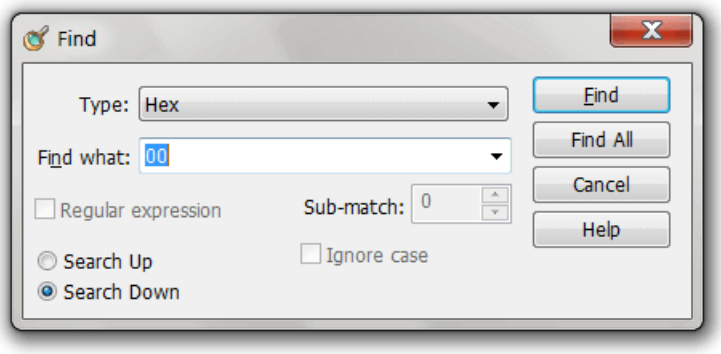
- If the Code Pane is active, numbers displayed in code pane are placed into the Clipboard. Settings like **Display As**, **Group By** and others affect the result.
- If the **Text Pane** is active, the textual representation of the selected data is placed into the Clipboard. Currently selected encoding for the view is used to convert text.

Exporting data

Executing the **Edit » Export...** command copies the selected binary data to the given file. Data is copied "as is".

Searching for Data

Raw Data View visualizer provides you with rich text and data searching capabilities. To start searching, execute the **Edit » Find...** command.



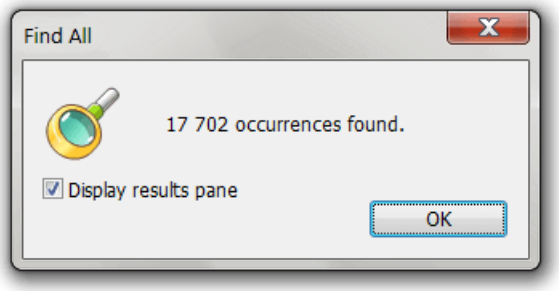
You may search for an encoded text string, UNICODE text string, regular expression or byte sequence. Select the type from the *Type* drop down box, enter the pattern to search in the *Find what* box, specify additional options and press either the **Find** or **Find All** buttons.

For both string options, you may enable regular expression mode. If regular expression mode is enabled, you may choose the sub-match to use. See the Regular Expressions topic for more information on supported syntax and options.

After you press the **Find** button, Device Monitoring Studio locates the first occurrence of a pattern. If pattern is not found, a message box is displayed. Use the **Edit » Find Next** command to continue searching.

Another option is to press the **Find All** button. In this mode, Device Monitoring Studio will search for all occurrences of a pattern and produce a multiple selection, containing all located ranges.

After you execute the Find All command, the Find All Results window is displayed:



If you leave the *Display results pane* box checked, the **Selection** tool window is displayed. It displays brief information about a multiple selection object, as well as a list of all ranges in a selection object. Clicking on a range moves the cursor to the beginning of the range. Right clicking on it opens a shortcut menu with following commands:

Go to Start

Move the cursor to the beginning of the range.

Go to End

Move the cursor to the end of the range.

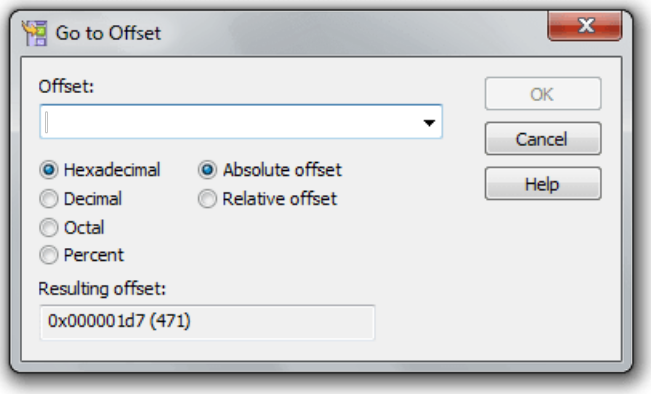
Delete Block

Removes the given range from a selection.

Clear Selection

Clears the selection.

Go to Offset



You can change the way entered offset is interpreted, according to the following table:

Number Type	Absolute offset	Relative offset
Hexadecimal	Absolute offset in hexadecimal format. Only non-negative values are allowed.	Relative offset in hexadecimal format. May either be positive or negative number. The resulting offset is then calculated by adding the given offset to the current one.
Decimal	Absolute offset in decimal format. Only non-negative values are allowed.	Relative offset in decimal format. May either be positive or negative number. The resulting offset is then calculated by adding the given offset to the current one.
Octal	Absolute offset in octal format. Only non-negative values are allowed.	Relative offset in octal format. May either be positive or negative number. The resulting offset is then calculated by adding the given offset to the current one.
Percent	Absolute offset in decimal format as a percent of the current file's size.	

The resulting offset is displayed at the bottom of the window in hexadecimal and decimal formats.

Regular Expressions

Regular expressions (RegEx) provide a concise and flexible means for identifying strings of text of interest, such as particular characters, words, or patterns of characters. Regular expressions are written in a formal language. The syntax used by the Device Monitoring Studio is essentially the syntax standardized by ECMAScript, with minor changes in support of internationalization. The following section briefly describes the syntax.

Regular expressions are supported by the following Raw Data View visualizer's features:

- Pattern Highlighting

Using this feature you may easily search across the monitored USB, Serial or Network data stream for specific packets/data patterns, which match regular expression (RegExp) entered in “Find what:” input field.

Capturing Sub-expressions

Regular expression matching allows you to specify what sub-expression you want to capture. Sub-

expression zero represents the entire expression, sub-expression 1 and greater represent corresponding sub-expressions.

If you specify the sub-expression that is greater than the total number of sub-expressions, the error message is displayed.

Usage Tips and Performance Considerations

Using regular expressions affect performance and memory usage. Always prefer using normal pattern searching functions whenever possible.

The following limitations are present in the current version of the Device Monitoring Studio:

- Searching for a regular expression *backwards* is not supported.
- Searching within selection (either single or multiple) is not supported.
- Pattern Highlighting will sometimes fail to highlight a complex match if it starts long before the visible area and/or ends long after the visible area.
- Zero-matching regular expressions are forbidden.

Regular Expressions Syntax

A regular expression is a pattern of text that consists of ordinary characters (for example, letters `a` through `z`) and special characters, known as *metacharacters*. The pattern describes one or more strings to match when searching text. The following table contains the complete list of *metacharacters* and their behavior in the context of regular expressions (regular expressions syntax):

Metacharacter	Description
`\`	Marks the next character as a special character, a literal, a backreference, or an octal escape. For example, `n` matches the character `n`. `\n` matches a newline character. The sequence `\\` matches `\` and `\(` matches `(`.
`^`	Matches the position at the beginning of the input string. Also matches the position following `\n` or `\r`.
`$`	Matches the position at the end of the input string. Also matches the position preceding `\n` or `\r`.
`*`	Matches the preceding character or sub-expression zero or more times. For example, `zo*` matches `z` and `zoo`. `*` is equivalent to `{0,}`.
`+`	Matches the preceding character or sub-expression one or more times. For example, `zo+` matches `zo` and `zoo`, but not `z`. `+` is equivalent to `{1,}`.
`?`	Matches the preceding character or sub-expression zero or one time. For example, `do(es)?` matches the `do` in `do` or `does`. `?` is equivalent to `{0,1}`.
`{n}`	`n` is a non-negative integer. Matches exactly `n` times. For example, `o{2}` does not match the `o` in `Bob`, but matches the two o's in `food`.
`{n,}`	`n` is a non-negative integer. Matches at least `n` times. For example, `o{2,}` does not match the `o` in `Bob` and matches all the o's in `foooood`. `o{1,}` is equivalent to `o+`. `o{0,}` is equivalent to `o*`.

Metacharacter	Description
`{n,m}`	`m` and `n` are non-negative integers, where `n` <= `m`. Matches at least `n` and at most `m` times. For example, `o{1,3}` matches the first three o's in `fooooood`. `o{0,1}` is equivalent to `o?`. Note that you cannot put a space between the comma and the numbers.
`?`	When this character immediately follows any of the other quantifiers (`*`, `+`, `?`, `{n}`, `{n,}`, `{n,m}`), the matching pattern is non-greedy. A non-greedy pattern matches as little of the searched string as possible, whereas the default greedy pattern matches as much of the searched string as possible. For example, in the string `oooo`, `o+?` matches a single `o`, while `o+` matches all `o`s.
`.`	Matches any single character.
`(pattern)`	A sub-expression that matches pattern and captures the match. To match parentheses characters ( ), use `\(` or `\)`.
`(?:pattern)`	A sub-expression that matches pattern but does not capture the match, that is, it is a non-capturing match that is not stored for possible later use. This is useful for combining parts of a pattern with the "or" character `|`. For example, `industr(?:y|ies)` is a more economical expression than `industry|industries`.
`(?=pattern)`	A sub-expression that performs a positive lookahead search, which matches the string at any point where a string matching pattern begins. This is a non-capturing match, that is, the match is not captured for possible later use. For example `Windows (?=95|98|NT|2000)` matches `Windows` in `Windows 2000` but not `Windows` in `Windows 3.1`. Look-aheads do not consume characters, that is, after a match occurs, the search for the next match begins immediately following the last match, not after the characters that comprised the lookahead.
`(?!pattern)`	A sub-expression that performs a negative lookahead search, which matches the search string at any point where a string not matching pattern begins. This is a non-capturing match, that is, the match is not captured for possible later use. For example `Windows (?!95|98|NT|2000)` matches `Windows` in `Windows 3.1` but does not match `Windows` in `Windows 2000`. Look-aheads do not consume characters, that is, after a match occurs, the search for the next match begins immediately following the last match, not after the characters that comprised the lookahead.
`(?>pattern)`	Independent sub-expression, match pattern and turn off backtracking.
`(?<=pattern)`	Positive look-behind assertion, match if after pattern but don't include pattern in the match. (pattern must be constant-width).
`(?<!pattern)`	Negative look-behind assertion, match if not after pattern. (pattern must be constant-width).
`(?i:pattern)`	Match pattern disregarding case. Allows to temporary turn off case sensitiveness during pattern matching.
`(?$name)`	Reference a named regular expression class. A class must be defined in the Regular Expressions Settings, otherwise, the error will be generated.
`x|y`	Matches either `x` or `y`. For example, `z|food` matches `z` or `food`. `(z|f)ood` matches `zood` or `food`.

Metacharacter	Description
`[xyz]`	A character set. Matches any one of the enclosed characters. For example, `[abc]` matches the `a` in `plain`.
`[^xyz]`	A negative character set. Matches any character not enclosed. For example, `[^abc]` matches the `p` in `plain`.
`[a-z]`	A range of characters. Matches any character in the specified range. For example, `[a-z]` matches any lowercase alphabetic character in the range `a` through `z`.
`[^a-z]`	A negative range characters. Matches any character not in the specified range. For example, `[^a-z]` matches any character not in the range `a` through `z`.
`\b`	Matches a word boundary, that is, the position between a word and a space. For example, `er\b` matches the `er` in `never` but not the `er` in `verb`.
`\B`	Matches a non-word boundary. `er\B` matches the `er` in `verb` but not the `er` in `never`.
`\cx`	Matches the control character indicated by `x`. For example, `\cM` matches a Control-M or carriage return character. The value of `x` must be in the range of `A-Z` or `a-z`. If not, `c` is assumed to be a literal `c` character.
`\d`	Matches a digit character. Equivalent to `[0-9]`.
`\D`	Matches a non-digit character. Equivalent to `[^0-9]`.
`\f`	Matches a form-feed character. Equivalent to `\x0c` and `\cL`.
`\n`	Matches a newline character. Equivalent to `\x0a` and `\cJ`.
`\r`	Matches a carriage return character. Equivalent to `\x0d` and `\cM`.
`\s`	Matches any white space character including space, tab, form-feed, and so on. Equivalent to `[ \f\n\r\t\v]`.
`\S`	Matches any non-white space character. Equivalent to `[^ \f\n\r\t\v]`.
`\t`	Matches a tab character. Equivalent to `\x09` and `\cI`.
`\v`	Matches a vertical tab character. Equivalent to `\x0b` and `\cK`.
`\w`	Matches any word character including underscore. Equivalent to `[A-Za-z0-9_]`.
`\W`	Matches any non-word character. Equivalent to `[^A-Za-z0-9_]`.
`\xn`	Matches `n`, where `n` is a hexadecimal escape value. Hexadecimal escape values must be exactly two digits long. For example, `\x41` matches `A`. `\x041` is equivalent to `\x04` & `1`. Allows ASCII codes to be used in regular expressions.
`\num`	Matches `num`, where `num` is a positive integer. A reference back to captured matches. For example, `(.)\1` matches two consecutive identical characters.
`\n`	Identifies either an octal escape value or a back-reference. If `\n` is preceded by at least `n` captured sub-expressions, `n` is a back-reference. Otherwise, `n` is an octal escape value if `n` is an octal digit (0-7).

Metacharacter	Description
`\nm`	Identifies either an octal escape value or a back-reference. If `\nm` is preceded by at least `nm` captured sub-expressions, `nm` is a back-reference. If `\nm` is preceded by at least `n` captures, `n` is a back-reference followed by literal `m`. If neither of the preceding conditions exists, `\nm` matches octal escape value `nm` when `n` and `m` are octal digits (0-7).
`\nml`	Matches octal escape value `nml` when `n` is an octal digit (0-3) and `m` and `l` are octal digits (0-7).
`\un`	Matches `n`, where `n` is a Unicode character expressed as four hexadecimal digits. For example, `\u00A9` matches the copyright symbol (©).

Examples

This section lists a few regular expressions examples. The description section describes the results of the **Find All** command used with a given regular expression.

`\w+`

Match all identifiers in a programming language source file.

`(.)\1+`

Match any sequence of repeated characters, which length is at least two characters.

`[\x00\x01\x02]`

Match 00, 01 or 02 bytes in the document.

`\w+(?=\s*\*)`

Match all C++ class names in the document which are followed by a `*` character. `*` character and any optional space characters before it are not matched.

`(?<=http://)[\w\.]+?\.(com|net|org)`

Match all domain name portions of URLs.

`\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\`

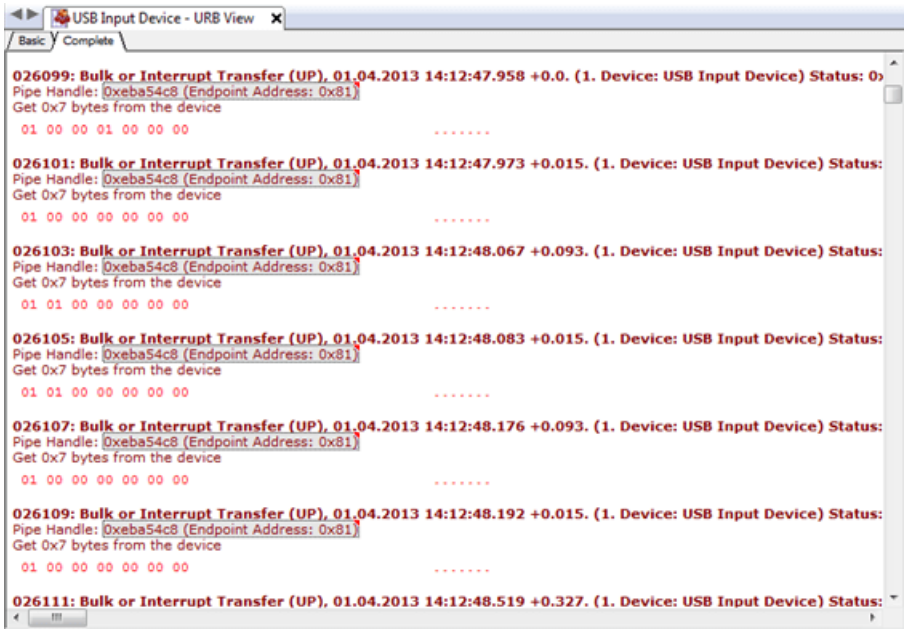
Match all IPv4 addresses.

`^{`

Match “{” characters at the beginning of the line.

URB View

This visualizer decodes USB Request Block (URB) packets. It also displays all PnP packets. All monitored URBs can be displayed in one of two schemes - Basic or Complete. In Basic scheme, only general information about a packet is displayed, while in Complete scheme full packet information, including the binary data attached to the packet is displayed.



For each captured packet the following information is displayed:

Packet number

This is a consecutive packet number. This number is session-wide and will be the same in each session's visualizer.

Packet type

Packet type such as “Bulk or Interrupt Transfer” or “Control Transfer”.

(Direction)

Can be either **DOWN** or **UP**. **DOWN** means that the packet is captured on its way down, that is, from controlling application through device's driver to USB host controller and finally, to device. **UP** means the opposite direction. Each packet is always captured twice: first on its way down and then on its way up. Default filtering settings may eventually hide one of these captures.

Packet capture time

Packet capture time in current user's full date/time format.

- **Difference in seconds**

  Difference in seconds between the current packet and the previous packet. Note that the previous packet is the packet with a previous consecutive number. A previous packet may be missing in this and/or other data visualizer.

(N. Device Name)

An ordinal number and device name for multi-source sessions.

Status: 0xNNNNNNNN

Internal NT status code. Zero means success.

URB View visualizer supports Generic Filtering platform.

Exporting Data

URB View visualizer allows the user to export its contents to various text file formats. First, you need to determine which part of the window you need to export.

If you plan to export the whole contents of the window, make sure you have **no selection**. Otherwise,

use the mouse to select a portion of the window.

Execute the **Edit » Copy** command to copy the selected content to the Clipboard. Execute the **Edit » Export...** command to export the selected content to a file. In the latter case, you may select the format to use during exporting:

HTML Files

Export the content in HTML format. This format saves all the original formatting.

ASCII Text Files

Copy of the window content in text format in ASCII encoding.

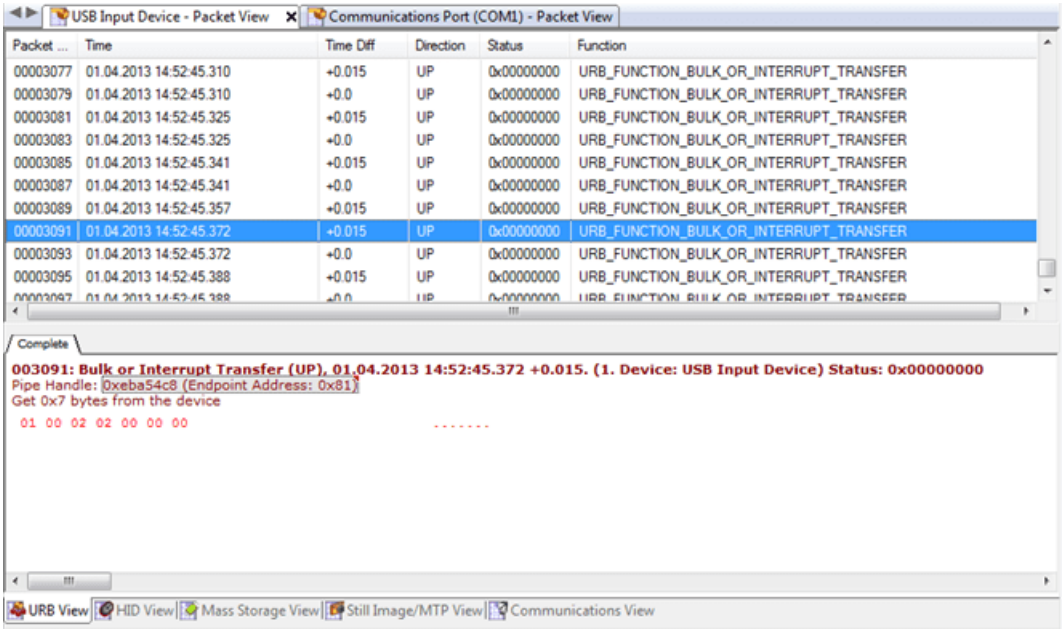
UNICODE Text Files

Copy of the window content in text format in UNICODE (UTF-16 LE) encoding.

In addition, use the **Edit » Export Binary** command to export the content of the individual packet in binary format.

Packet View

This visualizer window consists of two parts. The top part displays monitored packets in a table. For each packet, the following information is displayed:



Ordinal

This is a consecutive packet number. This number is session-wide and will be the same in each session's visualizer.

Time

Packet Capture Time

Time Diff

Difference in seconds between the current packet and the previous packet. Note that the previous packet is the packet with a previous consecutive number. A previous packet may be missing in this and/or other data visualizer.

Direction

Can be either **DOWN** or **UP**. **DOWN** means that the packet is captured on its way down, that is, from controlling application through device's driver to USB host controller and finally, to device. **UP**

means the opposite direction. Each packet is always captured twice: first on its way down and then on its way up. Default filtering settings may eventually hide one of these captures.

Status

Internal NT status code. Zero means success.

Function

Packet type

Device

For multi-source sessions, this column shows the ordinal number and name of the source device.

The bottom part is built from several visualizers, depending on session type. For USB, URB View, HID View, Mass Storage View, Still Image View and Communications View visualizers are present. For Serial, Request View, MODBUS Viewand PPP Viewvisualizers are present. Click on any packet in the top part to see it decoded by each visualizer in the bottom part. Click on the tab to switch to the visualizer.

Packet View visualizer supports Generic Filtering platform.

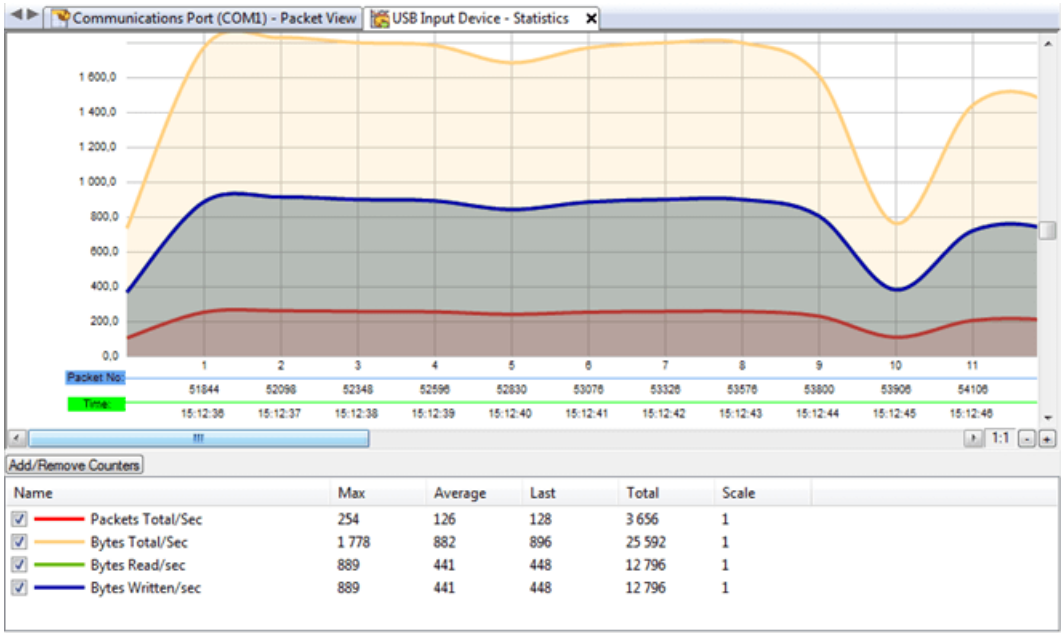
Exporting Data

Packet View visualizer allows the user to export its contents to a text format. First, you need to select packets you want to export. Use standard mouse or keyboard commands to select packets in a list or use the **Edit » Select All** command to select all packets.

Execute the **Edit » Copy** command to copy the selected content to the Clipboard. Execute the **Edit » Export...** command to export the selected content to a file. The format of exported data is so-called *Tab-Separated Values*, or *TSV*, which may be imported directly to spreadsheet applications, like Microsoft Excel.

Statistics

This visualizer collects and displays various data statistics on monitored stream. A list of counters depend on the session type.



USB

- Packets Total/Sec

- Packets Read/Sec
- Packets Written/Sec
- Bytes Total/Sec
- Bytes Read/Sec
- Bytes Written/Sec
- Control: Bytes Total/Sec
- Control: Bytes Read/Sec
- Control: Bytes Written/Sec
- Bulk: Bytes Total/Sec
- Bulk: Bytes Read/Sec
- Bulk: Bytes Written/Sec
- Interrupt: Bytes Total/Sec
- Interrupt: Bytes Read/Sec
- Interrupt: Bytes Written/Sec
- Isochronous: Bytes Total/Sec
- Isochronous: Bytes Read/Sec
- Isochronous: Bytes Written/Sec

Serial

- Packets Total/Sec
- Bytes Total/Sec
- Bytes Read/Sec
- Bytes Written/Sec
- IO Packets/Sec

Network

- Packets/Sec
- Bytes Total/Sec
- Bytes Received/Sec
- Bytes Sent/Sec

By default, only some of these counters are displayed. To add more, click the **Add/Remove Counter** button.

Bottom part of the visualizer displays the variable statistics. For each variable, the following is displayed:

- Variable Name, line width and color, visibility flag. Click on the checkbox to toggle variable on or off.
- Maximum value
- Average value
- Last value
- Total
- Scale

You can toggle each variable visibility by clicking on the checkbox next to variable's name. You can also change the variable's color, line width and scale by double-clicking the variable to display the **Variable Properties** window.

Click anywhere on the plot area to place a check mark (a so-called Static Line). After you place a static line, you will see the momentary values for each variable at the top-right corner of the visualizer.

If you hold a `Ctrl` key while clicking on the plot area, you place a tracking line, which is bound to the data and does not move.

To remove either the static or the tracking line, click on the far right part of the plot area, just under the lines' informational area.

The visualizer displays three different kinds of data on the X-axis, sample number, packet number and time mark. You can use the *Display sample indices on X-axis* option on Tools » Settings, General Tab to toggle displaying the sample numbers on X-axis.

Advanced

Adjusting Output

You can change the scale of individual variables to adjust their representation on the chart. Use the vertical scrollbar to change the Y-axis scale.

Use the following commands (available from context menu):

Auto Fit

When enabled, this option causes the plot to scale the visible data automatically so you always see the minimum and maximum values on the screen. Auto fitting concerns all enabled variables and may cause the Y-axis scale to change as you scroll or change the horizontal scale. The vertical scrollbar adjustment continues to work in this mode as well.

Draw Lines

The plot is drawn using polylines - this is the fastest method available.

Draw Curves

The plot is drawn using splines, effectively smoothing data. This method consumes more processor cycles and should be switched off if your computer becomes unresponsive. This method is automatically switched off on small scales.

No Fill

The area under the graphic is not filled - again, this is the fastest possible method. It is automatically chosen when you go to the smallest horizontal scales.

Solid Fill

The area under the graphic is filled with a solid color, being the slightly desaturated color of the variable itself.

Gradient Fill

The area under the graphic is filled with a gradient. This is the slowest method, so it should not be used if your computer becomes unresponsive.

The **Edit » Clear View** command deletes all data, including the statistics for each variable.

Navigating

Use the horizontal scrollbar to scroll the visible plot area. Use the + and - buttons to change the horizontal scale (from 1:64 to 1:1).

Use the **Edit » Go to Packet...** and **Go to Sample...** (from popup menu) commands to navigate to specific packet or sample correspondingly.

Capturing the Plot Data

You can save a copy of the visible data in the **Statistics** visualizer to the Clipboard or to the disk file. Use the **Edit » Copy** and **Edit » Export...** commands to copy the picture to Clipboard or file correspondingly. The picture copied to the Clipboard can be inserted into any application capable of working with bitmaps. **Edit » Export...** command allows you to save the picture in JPEG, PNG, GIF, BMP and TIFF formats.

Audio View

USB Audio visualizer parses packets and configuration descriptors for USB Audio (version 1.0) compliant device. The following subclasses are supported - Audio Control (AC) and Audio Streaming (AS). The first one is used to control and change the state of device. The second one is generally used for data transmitting. Audio class descriptors are displayed in Configuration Descriptor pane.

The following control pipe requests are parsed for the audio class device:

- SET_CUR (0x01)
- GET_CUR (0x81)
- SET_MIN (0x02)
- GET_MIN (0x82)
- SET_MAX (0x03)
- GET_MAX (0x83)
- SET_RES (0x04)
- GET_RES (0x84)
- SET_MEM (0x05)
- GET_MEM (0x85)
- GET_STAT (0xFF)

Audio View visualizer supports Generic Filtering platform.

You can select subclass (AudioControl or AudioStreaming) on Tools » Settings, General Tab page.

Exporting Data

Audio View visualizer allows the user to export its contents to various text file formats. First, you need to determine which part of the window you need to export.

If you plan to export the whole contents of the window, make sure you have **no selection**. Otherwise, use the mouse to select a portion of the window.

Execute the **Edit » Copy** command to copy the selected content to the Clipboard. Execute the **Edit » Export...** command to export the selected content to a file. In the latter case, you may select the format to use during exporting:

HTML Files

Export the content in HTML format. This format saves all the original formatting.

ASCII Text Files

Copy of the window content in text format in ASCII encoding.

UNICODE Text Files

Copy of the window content in text format in UNICODE (UTF-16 LE) encoding.

In addition, use the **Edit » Export Binary** command to export the content of the individual packet in binary format.

Video View

USB Video visualizer parses packets and configuration descriptors for USB Video (version 1.1) compliant device. The following subclasses are supported - Video Control (VC) and Video Streaming (VS). The first one is used to control and change the state of device. The second one is generally used for data transmitting. Video class descriptors are displayed in Configuration Descriptor pane.

Video View visualizer supports Generic Filtering platform.

You can select subclass (VideoControl or VideoStreaming) on Tools » Settings, General Tab page.

Exporting Data

Video View visualizer allows the user to export its contents to various text file formats. First, you need to determine which part of the window you need to export.

If you plan to export the whole contents of the window, make sure you have **no selection**. Otherwise, use the mouse to select a portion of the window.

Execute the **Edit » Copy** command to copy the selected content to the Clipboard. Execute the **Edit » Export...** command to export the selected content to a file. In the latter case, you may select the format to use during exporting:

HTML Files

Export the content in HTML format. This format saves all the original formatting.

ASCII Text Files

Copy of the window content in text format in ASCII encoding.

UNICODE Text Files

Copy of the window content in text format in UNICODE (UTF-16 LE) encoding.

In addition, use the **Edit » Export Binary** command to export the content of the individual packet in binary format.

HID View

This visualizer decodes Human Interface Device (HID) specific packets, displaying them in two schemes: HID View and Report View.

HID View visualizer supports Generic Filtering platform.

HID View

In this scheme, the visualizer displays parsed HID Report descriptor requests and brief description of each monitored HID packet.

Report View

In this scheme, the most detailed information for each monitored packet is displayed. For each variable the bounded string, usage information from the usage table, logical and physical boundary values as well as the variable value and measurement unit are displayed. Some fields may not be appropriate for some packets.

Exporting Data

HID View visualizer allows the user to export its contents to various text file formats. First, you need to determine which part of the window you need to export.

If you plan to export the whole contents of the window, make sure you have **no selection**. Otherwise, use the mouse to select a portion of the window.

Execute the **Edit » Copy** command to copy the selected content to the Clipboard. Execute the **Edit » Export...** command to export the selected content to a file. In the latter case, you may select the format to use during exporting:

HTML Files

Export the content in HTML format. This format saves all the original formatting.

ASCII Text Files

Copy of the window content in text format in ASCII encoding.

UNICODE Text Files

Copy of the window content in text format in UNICODE (UTF-16 LE) encoding.

In addition, use the **Edit » Export Binary** command to export the content of the individual packet in binary format.

Mass Storage View

This visualizer parses and displays commands and data exchanged by the computer and devices conforming to USB Mass Storage Bulk-Only device class and subclass. Specific commands from USB Mass Storage CBI Transport are also supported. There are two levels of display offered by the visualizer: Mass Storage Level and Command Level. The first one parses Mass Storage-specific structures, while the second one parses commands exchanged between the host and device.

Mass Storage standard allows using several command set, including SCSI Primary Command Set, SCSI Reduced Block Set and SCSI Multimedia Command Set. They are usually commonly referred as SCSI Transparent Command Set.

The visualizer always tries to process the command using the current command set (reported by the device). If it fails, it looks for the command in other supported command sets and parses it. In the latter case, you see the warning displayed next to the parsed command. If multiple command sets include definitions for the single command, it will be parsed using the first one that matches.

Supported Commands

The visualizer conforms to the following standards:

- SCSI Primary Command Set 2 (SPC2)
- SCSI Multimedia Command Set 2 (MMC2)
- SCSI Reduced Block Command Set (RBC)

Below you will find the list of supported commands. If the command has a number next to its name, that means there are several similar commands differing by the transfer length. The user has an option to select the commands the parser processes in the Tools » Settings, Filtering Tab.

MMC

- BLANK (0xa1)
- FORMAT_UNIT (0x04)
- CLOSE_TRACK_SESSION (0x5b)
- GET_CONFIGURATION (0x46)
- GET_PERFORMANCE (0xac)
- LOAD_UNLOAD_MEDIUM (0xa6)
- MECHANISM_STATUS (0xbd)
- PAUSE_RESUME (0x4b)
- PLAY_AUDIO10 (0x45)
- PLAY_AUDIO12 (0xa5)
- PLAY_AUDIO_MSF (0x47)
- READ_BUFFER_CAPACITY (0x5c)
- PLAY_CD (0xbc)
- READ_CD (0xbe)
- READ_CD_MSF (0xb9)

SPC2

- TEST_UNIT_READY (0x00)
- REQUEST_SENSE (0x03)
- INQUIRY (0x12)
- MODE_SENSE6 (0x1a)
- MODE_SENSE10 (0x5a)
- EXTENDED_COPY (0x83)
- LOG_SELECT (0x4c)
- LOG_SENSE (0x4d)
- MODE_SELECT6 (0x15)
- MODE_SELECT10 (0x55)
- PERSISTENT_RESERVE_IN (0x5e)
- PERSISTENT_RESERVE_OUT (0x5f)
- PREVENT_MEDIUM_REMOVAL (0x1e)
- READ_BUFFER (0x3c)
- RECIEVE_COPY_RESULTS (0x84)

- READ_CAPACITY (0x25)
- READ_DISC_INFORMATION (0x51)
- READ_DVD_STRUCTURE (0xad)
- READ_FORMAT_CAPACITIES (0x23)
- READ_HEADER (0x44)
- READ_MASTER_CUE (0x59)
- READ_SUB_CHANEL (0x42)
- READ_TOC (0x43)
- READ_TRACK_INFO (0x52)
- WRITE10 (0x2a)
- REPAIR_TRACK (0x58)
- REPORT_KEY (0xa4)
- RESERVE_TRACK (0x53)
- SCAN (0xba)
- SEND_CUE_SHEET (0x5d)
- SEND_DVD_STRUCTURE (0xbf)
- SEND_EVENT (0xa2)
- SEND_KEY (0xa3)
- SEND_CD_SPEED (0xbb)
- SEND_OPC_INFO (0x54)
- SET_READ_AHEAD (0xa7)
- SET_STREAMING (0xb6)
- STOP_PLAY_SCAN (0x4e)
- WRITE_AND_VERIFY (0x2e)
- READ12 (0xaa)
- ERASE10 (0x2c)
- GET_EVENT (0x4a)
- RECEIVE_DIAGNOSTIC_RESULTS (0x1c)
- RELEASE10 (0x57)
- RELEASE6 (0x17)
- REPORT_DEVICE_IDENTIFIER (0xa3)
- LUNS(0xa0)
- RESERVE10 (0x56)
- RESERVE6 (0x16)
- SEND_DIAGNOSTICS (0x1d)
- SET_DEVICE_IDENTIFIER (0xa4)
- WRITE_BUFFER (0x3b)

RBC

- READ10 (0x28)
- READ_CAPACITY (0x25)
- SYNC_CACHE (0x35)
- FORMAT_UNIT (0x04)
- START_STOP_UNIT (0x1b)
- WRITE10 (0x2a)
- VERIFY10 (0x2f)

Some devices may use newer command set revisions. In this case one of the following occurs:

- The parser will not be able to process the command correctly if its layout changed.
- The parser will correctly process the command, but without new fields added in the newer revision.
- The parser will not be able to process the command, not defined in the supported revision.

In all of three cases, the warning will be displayed.

Mass Storage View visualizer supports Generic Filtering platform.

Exporting Data

Mass Storage View visualizer allows the user to export its contents to various text file formats. First, you need to determine which part of the window you need to export.

If you plan to export the whole contents of the window, make sure you have **no selection**. Otherwise, use the mouse to select a portion of the window.

Execute the **Edit » Copy** command to copy the selected content to the Clipboard. Execute the **Edit » Export...** command to export the selected content to a file. In the latter case, you may select the format to use during exporting:

HTML Files

Export the content in HTML format. This format saves all the original formatting.

ASCII Text Files

Copy of the window content in text format in ASCII encoding.

UNICODE Text Files

Copy of the window content in text format in UNICODE (UTF-16 LE) encoding.

In addition, use the **Edit » Export Binary** command to export the content of the individual packet in binary format.

Still Image View

This visualizer decodes Still Image specific packets.

Still Image View visualizer parses and displays commands and data exchanged by the device and host over USB StillImage/MTP protocol. The parser supports standard PIMA 15740/ MTP commands. You can configure filtering on Tools » Settings, Filtering Tab page.

Still Image View visualizer supports Generic Filtering platform.

Exporting Data

Still Image View visualizer allows the user to export its contents to various text file formats. First, you need to determine which part of the window you need to export.

If you plan to export the whole contents of the window, make sure you have **no selection**. Otherwise, use the mouse to select a portion of the window.

Execute the **Edit » Copy** command to copy the selected content to the Clipboard. Execute the **Edit » Export...** command to export the selected content to a file. In the latter case, you may select the format to use during exporting:

HTML Files

Export the content in HTML format. This format saves all the original formatting.

ASCII Text Files

Copy of the window content in text format in ASCII encoding.

UNICODE Text Files

Copy of the window content in text format in UNICODE (UTF-16 LE) encoding.

In addition, use the **Edit » Export Binary** command to export the content of the individual packet in binary format.

Communications View

This visualizer decodes USB Communications Class packets and descriptors.

Communications View visualizer supports Generic Filtering platform.

Exporting Data

Communications View visualizer allows the user to export its contents to various text file formats. First, you need to determine which part of the window you need to export.

If you plan to export the whole contents of the window, make sure you have **no selection**. Otherwise, use the mouse to select a portion of the window.

Execute the **Edit » Copy** command to copy the selected content to the Clipboard. Execute the **Edit » Export...** command to export the selected content to a file. In the latter case, you may select the format to use during exporting:

HTML Files

Export the content in HTML format. This format saves all the original formatting.

ASCII Text Files

Copy of the window content in text format in ASCII encoding.

UNICODE Text Files

Copy of the window content in text format in UNICODE (UTF-16 LE) encoding.

In addition, use the **Edit » Export Binary** command to export the content of the individual packet in binary format.

Request View

This visualizer is based on Custom View data visualizer. It supports serial sessions and displays all monitored data packets and serial input/output control code packets.

Serial IOCTL packets, which are used to configure a serial device or retrieve its parameters are decoded and displayed in easy-to-use table.

Data packets are followed by the full data dump.

User Experience

See the section Custom View User Experience for more information on user interaction with **Request View** data visualizer.

Visual Schemes

Request View allows the user to customize the appearance of all its visual elements. Use the **Coloring** command from the context menu to jump to the Coloring customization section.

Configurable Options

The **Request View** defines the following options which control the amount of data the visualizer provides:

Option	Default Value	Description
Display empty reads	Off	Indicate whether the read requests that complete without data should be included.
Display I/O requests without data	Off	All I/O requests usually carry interesting information only on one way (either DOWN or UP, depending on the request). If this option is OFF, request with no data is not included in the output.
Display I/O requests	On	Indicate if serial IOCTL requests are included.
Display connect and create requests	On	Indicate whether connect and create requests are included.
Display down reads	Off	Normally, read requests are only displayed on their way UP, where they contain data retrieved from the device. Enabling this option will include read requests on their way DOWN as well. Note that failed read requests are always included.
Display up writes	Off	Normally, write requests are only displayed on their way DOWN, where they contain data to be sent to the device. Enabling this option will include write requests on their way UP as well. Note that failed write requests are always included.
Display line numbers	Off	Configures the display of line numbers for the visualizer.

Legacy Visualizer

The previous version of **Request View** data visualizer (which is not based on Custom View) is still available under the name Request View (Legacy).

Console View

This visualizer is based on Custom View data visualizer. It supports serial sessions and works as the text console, showing all transmitted and received data as ASCII characters.

Visual Schemes

Console View allows the user to customize the appearance of all its visual elements. Use the **Coloring** command from the context menu to jump to the Coloring customization section.

User Experience

See the section Custom View User Experience for more information on user interaction with Console View data visualizer.

Legacy Visualizer

The previous version of **Console View** data visualizer (which is not based on **Custom View**) is still

available under the name Console View (Legacy).

Data View

This visualizer contains two panes. The top pane displays what was read from the device, and the bottom pane displays the data written to the device.

Use the mouse to move the splitter to change the relative size of both panes.

Standard **Edit** commands, such as **Edit » Copy** and **Edit » Export...** are supported for both panes.

If the last line of information is visible, then the window is automatically scrolled every time a new portion of information is appended to it. To stop automatic scrolling, just scroll the window so the last line is not visible anymore. To resume automatic scrolling, scroll to the very end or just press the **End** key.

Either pane can have a selection. To make a selection use the mouse - click at the beginning of the selection and drag to the end. If you move the mouse away from the window during dragging, the window will scroll its contents. If the selection is present, the copy and export functions will work only on the selected part of the window.

Data View visualizer supports Generic Filtering platform.

Serial Bridge

When used with the Serial Bridge monitoring session type, the top pane displays the data the first bridged device sends and the bottom pane displays the data, which the second bridged device sends.

Exporting Data

Data View visualizer allows the user to export its contents to various text file formats. First, you need to determine which part of the window you need to export.

If you plan to export the whole contents of the window, make sure you have **no selection**. Otherwise, use the mouse to select a portion of the window.

Execute the **Edit » Copy** command to copy the selected content to the Clipboard. Execute the **Edit » Export...** command to export the selected content to a file. In the latter case, you may select the format to use during exporting:

HTML Files

Export the content in HTML format. This format saves all the original formatting.

ASCII Text Files

Copy of the window content in text format in ASCII encoding.

UNICODE Text Files

Copy of the window content in text format in UNICODE (UTF-16 LE) encoding.

In addition, use the **Edit » Export Binary** command to export the content of the individual packet in binary format.

Both panes of **Data View** visualizer are copied or exported separately.

MODBUS View

MODBUS is an application layer messaging protocol, positioned at level 7 of the OSI model that provides client/server communication between devices connected on different types of buses or networks.

The industry's serial de facto standard since 1979, MODBUS continues to enable millions of automation devices to communicate. Today, support for the simple and elegant structure of MODBUS continues to grow. The Internet community can access MODBUS at a reserved system port 502 on the TCP/IP stack.

The Modbus protocol has two different types of interaction: ASCII/RTU. In ASCII mode data is wrapped into the packet and transmitted as an ASCII string. The first byte of packet is used for identification – its value should be equal to `0x3A`. Packet ends with two special bytes - `0x0D` and `0x0A`.

MODBUS View visualizer is compliant with MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1a. There are several different implementations of Modbus Protocol. Different companies implement and document their own specific extensions to Modbus Protocol. Function and command names could vary: for example, Preset Single Register command in one implementation equals to Write Single Register command in the standard.

MODBUS View visualizer parses MODBUS protocol requests and responses, while the MODBUS Send Module can be used to construct and send MODBUS requests and responses. The visualizer is part of the monitoring infrastructure, which you can use it to monitor the existing connection between the application and a serial device. The **MODBUS Send Module** on the other hand, allows you to control the MODBUS-compliant device without the need of any third party controlling application. In this case, the **MODBUS View** visualizer, if configured, can be used to see the requests sent by the **MODBUS Send Module** and packets the device responds with.

You can set the filtering on Tools » Settings, Filtering Tab page.

The following settings govern the behavior of the visualizer (they all are controlled from the Tools » Settings, General Tab):

- Truncate register/coil/request list if it is too long - if there are many items in a request, only several first items will be displayed and others omitted.
- Add base offsets for registers, discrete inputs, etc. - in some implementations index of input register, holding register or discrete input is interpreted by a device like an offset. In this case, the value is appended to a so-called "base address" by the device. For example if you specify holding register 5 in your request, it can be parsed as register 40005 by a device (for a base address of 40000). The visualizer can be configured to account for this case using this option.
- Parse request on WRITE (responses on READ) direction – set this option for standard parsing – requests will be parsed as if they sent from host to device ("write" direction) and responses are parsed as if they are received from device ("read" direction). However, in some cases for debugging, it is useful to "revert" the flow. Deselect this option if you want request be parsed on the "read" way and responses on "write" way (host is responding to the device).
- Concatenate packets - set this option if packets are sent split (for example one packet with 8 bytes of payload is split into two packets: 2 + 6 bytes). The visualizer will try to concatenate blocks of data. Collected packet will be parsed and displayed when LRC/CRC of collected data is valid.
- RTU mode - set this option if you have enabled "Concatenate packets" option (see above) and is using RTU mode. Visualizer can't automatically determine the mode when packets are sent split. When "Concatenate packets" option is disabled, this option doesn't affect the visualizer behavior.

MODBUS View visualizer supports Generic Filtering platform.

Exporting Data

MODBUS View visualizer allows the user to export its contents to various text file formats. First, you need to determine which part of the window you need to export.

If you plan to export the whole contents of the window, make sure you have **no selection**. Otherwise, use the mouse to select a portion of the window.

Execute the **Edit » Copy** command to copy the selected content to the Clipboard. Execute the **Edit » Export...** command to export the selected content to a file. In the latter case, you may select the format to use during exporting:

HTML Files

Export the content in HTML format. This format saves all the original formatting.

ASCII Text Files

Copy of the window content in text format in ASCII encoding.

UNICODE Text Files

Copy of the window content in text format in UNICODE (UTF-16 LE) encoding.

In addition, use the **Edit » Export Binary** command to export the content of the individual packet in binary format.

PPP View

The Point-to-Point Protocol (PPP) provides a method for transmitting datagrams over serial point-to-point links. PPP is comprised of three main components:

1. A method for encapsulating datagrams over serial links.
2. A Link Control Protocol (LCP) for establishing, configuring, and testing the data-link connection.
3. A family of Network Control Protocols (NCPs) for establishing and configuring different network-layer protocols.

The **PPP View** visualizer is used to parse and display the PPP packets (such as requests, responses, options etc.).

PPP packet may have encapsulated frames with data inside. It is called "payload". Payload is formed using different protocols. There are several predefined protocols (such as LCP/ Novell IPX/ AppleTalk etc.).

The current version of the visualizer supports the following protocols:

1. Novell IPX (0x002b)
2. OSI Network Layer Control Protocol (0x8023)
3. DECnet Phase IV Control Protocol (0x8027)
4. Appletalk Control Protocol (0x8029)
5. Link Control Protocol (0xc021)
6. Password Authentication Protocol (0xc023)
7. Link Quality Report (0xc025)
8. Challenge Handshake Authentication Protocol (0xc223)
9. Multilink Protocol (0x003d)
10. Compression Control Protocol (0x80fd)
11. Extensible Authentication Protocol (0xc227)
12. NETBIOS Protocol (0x03f)

PPP View visualizer supports Generic Filtering platform.

Exporting Data

PPP View visualizer allows the user to export its contents to various text file formats. First, you need to determine which part of the window you need to export.

If you plan to export the whole contents of the window, make sure you have **no selection**. Otherwise, use the mouse to select a portion of the window.

Execute the **Edit » Copy** command to copy the selected content to the Clipboard. Execute the **Edit » Export...** command to export the selected content to a file. In the latter case, you may select the format to use during exporting:

HTML Files

Export the content in HTML format. This format saves all the original formatting.

ASCII Text Files

Copy of the window content in text format in ASCII encoding.

UNICODE Text Files

Copy of the window content in text format in UNICODE (UTF-16 LE) encoding.

In addition, use the **Edit » Export Binary** command to export the content of the individual packet in binary format.

Line View

This visualizer displays the state of the serial/modem control lines. The following lines states are displayed:

- RTS Request to Send line indicator
- CTS Clear to Send line indicator
- DSR Data Send Ready line indicator
- DCD Data Carrier Detect line indicator
- DTR Data Terminal Ready line indicator
- RI Ring line indicator

The gray circle shows that the line state is not determined at this moment. The red circle shows that the line is at low level and green circle shows the line is at high level.

Visualizer Positioning

This visualizer is unlike all others in the way it positions itself on the screen. Line states are displayed in the program status bar. In addition, you cannot configure more than one **Line View** visualizer for each monitoring session. When there are multiple monitoring sessions running, **Line View** visualizer displays line states for the currently selected monitoring session.

Request View (Legacy)

This visualizer displays all monitored packets, including PnP packets, data transfer packets and serial input/output control code packets. All monitored data can be displayed in one of two schemes - Basic or Complete. In Basic scheme, only general information about a packet is displayed, while in Complete scheme full packet information, including the binary data attached to the packet is displayed.

If the last line of information is visible, then the window is automatically scrolled every time a new portion of information is appended to it. To stop automatic scrolling, just scroll the window so the last line is not visible anymore. To resume automatic scrolling, scroll to the very end or just press the **End** key.

Request View (Legacy) visualizer supports Generic Filtering platform.

Exporting Data

Request View (Legacy) visualizer allows the user to export its contents to various text file formats. First, you need to determine which part of the window you need to export.

If you plan to export the whole contents of the window, make sure you have **no selection**. Otherwise,

use the mouse to select a portion of the window.

Execute the **Edit » Copy** command to copy the selected content to the Clipboard. Execute the **Edit » Export...** command to export the selected content to a file. In the latter case, you may select the format to use during exporting:

HTML Files

Export the content in HTML format. This format saves all the original formatting.

ASCII Text Files

Copy of the window content in text format in ASCII encoding.

UNICODE Text Files

Copy of the window content in text format in UNICODE (UTF-16 LE) encoding.

In addition, use the **Edit » Export Binary** command to export the content of the individual packet in binary format.

Console View (Legacy)

This visualizer works as the text console, showing all the received data as ASCII characters.

HTTP View

This visualizer is based on Custom View data visualizer. It supports network sessions and parses HTTP protocol requests and responses.

User Experience

See the section Custom View User Experience for more information on user interaction with **HTTP View** data visualizer.

Visual Schemes

HTTP View allows the user to customize the appearance of all its visual elements. Use the **Coloring** command from the context menu to jump to the Coloring customization section.

Data Recording

Data recording allows you to capture all (or part) of monitored data to the disk file for subsequent analysis. This section tells you how to create log files. The Playback section tells you how to play back previously recorded log files. The Tools » Settings, Recording/Playback Tab section describes Data Recording and Playback configuration options.

Log File Structure

Device Monitoring Studio log files have extension `.dmslog8`. New log is automatically started each time you add **Data Recording** processing module to the monitoring session. Each log consists of one or more files. Device Monitoring Studio supports the following logging modes:

Unlimited Mode

In unlimited mode all data is being written to a single log file and is never deleted. The maximum supported log size in this mode is 159.92 GB. This is the default mode.

WARNING

If your file grows up to the maximum limit, a multiple-files limited mode is automatically turned on and new part file is created.

Limited Modes

In addition to a single-file unlimited mode described above, Device Monitoring Studio supports single-file and multiple-files limited modes.

You may specify either the file size limit or a logging duration limit. Whenever the specified limit is reached, the following happens:

- In single-file limited mode some of the oldest data gets deleted and new data is getting written on top of it.
- In multiple-files limited mode, a current file (called a log part file) is closed and new log part file is created. Optionally, the oldest part file may also be deleted.

NOTE

In single-file limited mode, the minimum allowed file size is 80 MB. The limits you specify will not be applied until the log grows higher than 80 MB.

Configuring Data Recording

Data recording is taking place whenever a **Data Recording** processing module is added to a monitoring session. If you remove this processing module from a running session, recording stops; if you add this module to a running session, recording starts to a new log file. You cannot add more than one **Data Recording** module to a session.

For convenience, the following commands may be used as alternative:

Tools » Start/Resume Recording

Adds a **Data Recording** processing module to the current monitoring session or resumes a paused data recording.

Tools » Pause Recording

Pauses current recording.

Tools » Stop Recording

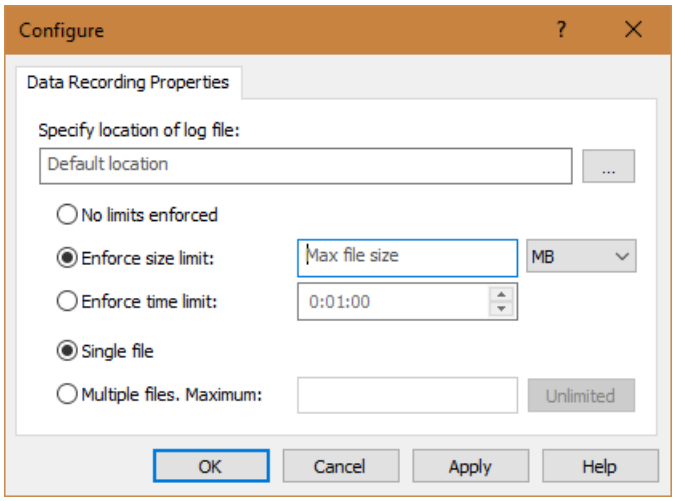
Removes a **Data Recording** processing module from the current monitoring session.

If you want to record new monitoring session from the very start, add the **Data Recording** processing module when you configure new monitoring session in Session Configuration Window.

Data Recording Options

Data Recording uses the log folder to store all log files it creates. This folder may be changed on the Tools » Settings, Recording/Playback Tab settings page.

In addition, a log file path may be changed for an individual session using **Data Recording** processing module configuration page:



Enter the name of the log file or leave the field empty to use the default.

Other options allow you to set the optional log file limits:

No limits enforced

The logging will work in single-file unlimited mode.

Enforce size limit

Enter the maximum size of a single part file.

Enforce time limit

Enter the maximum length of a single part file.

If size or time limit is specified, you can also specify if logging uses single-file limited mode or multiple-files limited mode:

Single file

Whenever specified limit is reached, new data override oldest data stored in a single log file.

Multiple files

Whenever specified limit is reached, a new part file is created and logging continues to the new file. You can also specify the maximum amount of part files to keep or leave it empty to keep all part files.

Save to Log

This option allows you to save all monitored data to the log file. For example, you monitored a device for a while without writing a log file and then decided to write all monitored data to a log for subsequent analysis.

Execute the **Tools » Save to Log** command and configure an output log file as described in the Recording Options topic.

> **WARNING**
>
> Amount of data available for saving depends on the current settings in the Tools » Settings, Data Processing Tab.

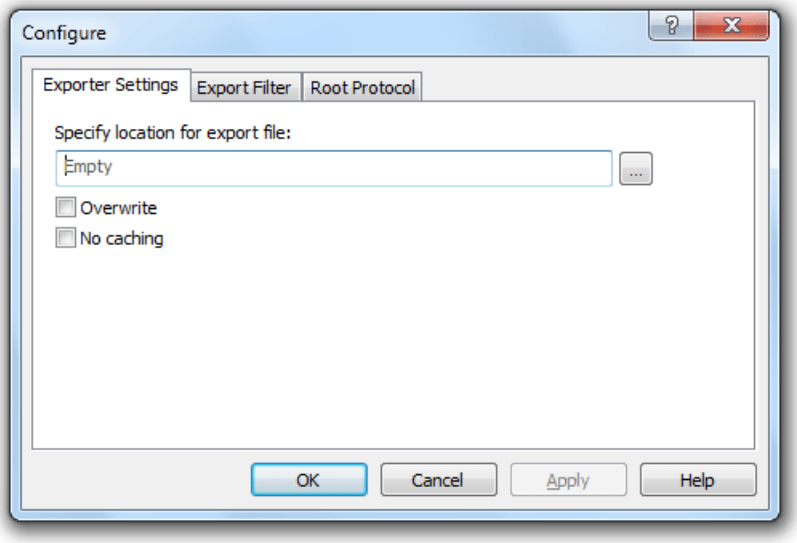
Raw Exporter

Raw Exporter data processing module works like lower part of the Structure View data visualizer, but

without displaying any information on the screen. Use this exporter when you need to parse monitored data according to some predefined or custom protocol, apply custom filters and copy the resulting binary packet data to the binary file.

Configuring Raw Exporter

This page allows you to specify location of the output binary log file.



Enter the location of the output log file or click the **Browse** button. Select **Overwrite** option to force overwriting the file and select **No caching** option to disable OS write caching (greatly affects performance).

Export Filter

See the Display Filter settings for more binary logging configuration info.

Root Protocol

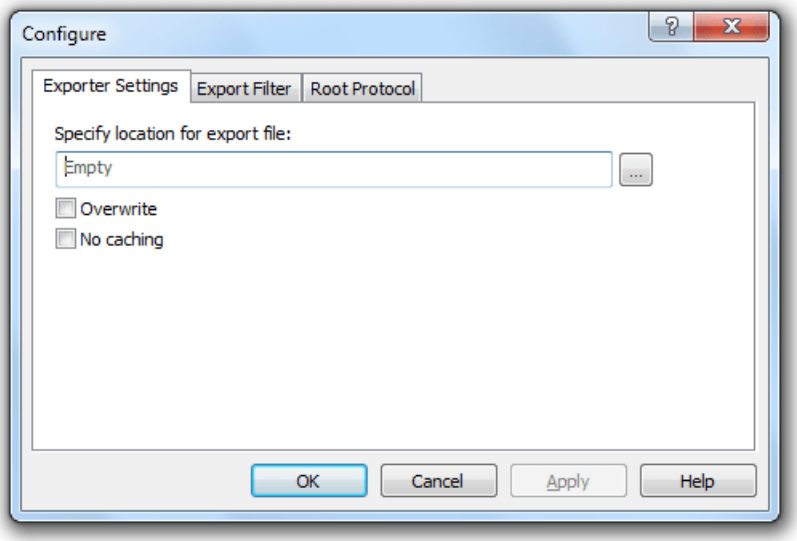
See the Root Protocol settings for more binary logging configuration info.

Text Exporter

Text Exporter data processing module works like upper part of the Structure View data visualizer, but without displaying any information on the screen. Use this exporter when you need to parse monitored data according to some predefined or custom protocol, apply custom filters and copy the resulting binary packet data to the text file.

Configuring Text Exporter

This page allows you to specify location of the output text log file.



Enter the location of the output file or click the **Browse** button. Select **Overwrite** option to force overwriting the file and select **No caching** option to disable OS write caching (greatly affects performance).

Export Filter

See the Display Filter settings for more text logging configuration info.

Root Protocol

See the Root Protocol settings for more text logging configuration info.

Advanced

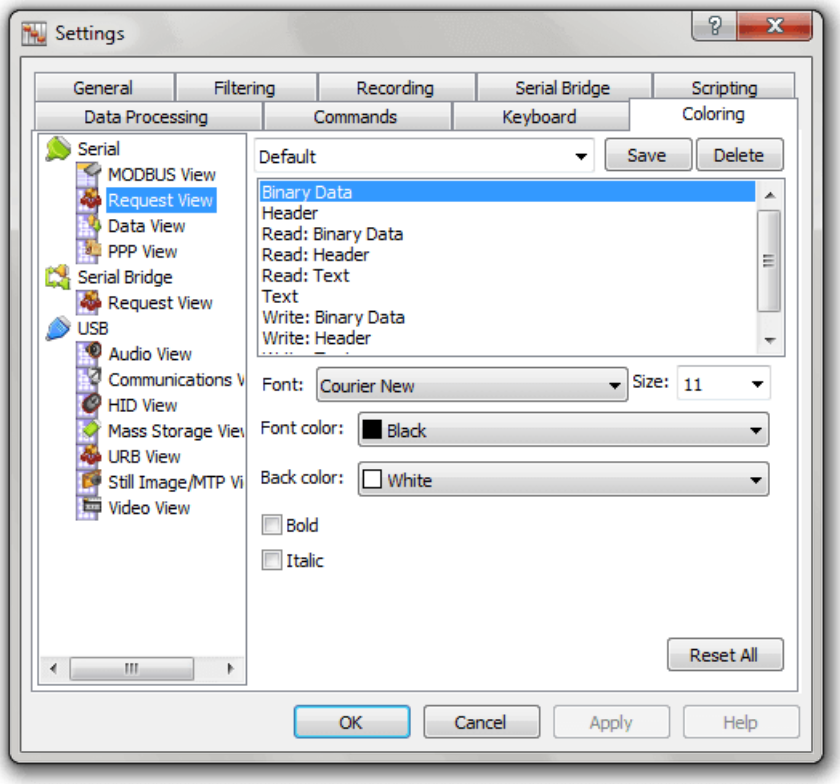
Generic Coloring

Device Monitoring Studio provides a generic mechanism that can be used to change the appearance of different visual elements of some data visualizers.

Each data visualizer that supports generic coloring defines a set of items for which you may set up font, foreground and background color.

You can configure visual element appearance and coloring schemes in the Tools » Settings, Coloring Tab.

Coloring Tab



The list of all supported data visualizers is displayed on the left. Select a data visualizer to configure the appearance of its visual elements.

After you select a visualizer, you will see the list of all its visual elements on the right of the window. For each element, you can specify the font face, font size and style as well as foreground and background color.

Working with Schemes

A scheme is a collection of all appearance settings for a given visualizer. You can select the scheme from the combobox at the top of the page. A scheme named "Default" is automatically created for every data visualizer and allows you to quickly return to the original appearance settings.

You can create your own schemes. To create a scheme, type the scheme name in the combobox and press the **Save** button. To delete a scheme, select a scheme from the list and press the **Delete** button. Note that you cannot delete a built-in scheme.

Data Recording (Previous version)

Data recording allows you to capture all (or part) of monitored data to the disk file for subsequent analysis. This section tells you how to create log files. The Playback section tells you how to play back previously recorded log files. The Tools » Settings, Recording/Playback Tab section describes Data Recording and Playback configuration options.

Log File Structure

Every single log file consists of several streams. Each time you start a monitoring session with **Legacy Data Recording** processing module, new log file is created for it. Sessions Tool Window lists all data processing modules for a session, including the **Legacy Data Recording**. The *State* column shows the total amount of data saved to a log file in bytes. It also allows you to press the **End Stream** button to stop current stream. The button changes to **New Stream** after that. Press it again to start new stream

within the same log file.

Configuring Data Recording

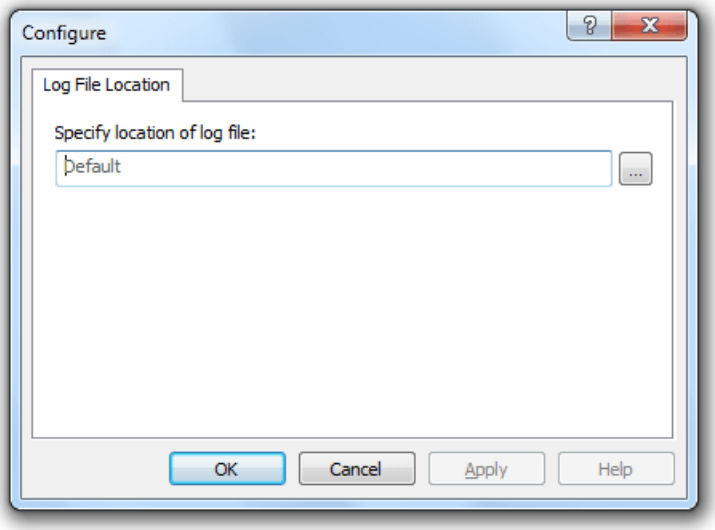
Data recording is taking place whenever a **Legacy Data Recording** processing module is added to a monitoring session. If you remove this processing module from a running session, recording stops; if you add this module to a running session, recording starts to a new log file. You cannot add more than one **Legacy Data Recording** module to a session.

If you want to record new monitoring session from the very start, add the **Legacy Data Recording** processing module when you configure new monitoring session in Session Configuration Window.

Configuring Recording Options

Legacy Data Recording uses the log folder to store all log files it creates. This folder may be changed on the Tools » Settings, Recording/Playback Tab settings page.

In addition, a log file path may be changed for an individual session using **Legacy Data Recording** processing module configuration page:



Enter the name of the log file or leave the field empty to use the default.

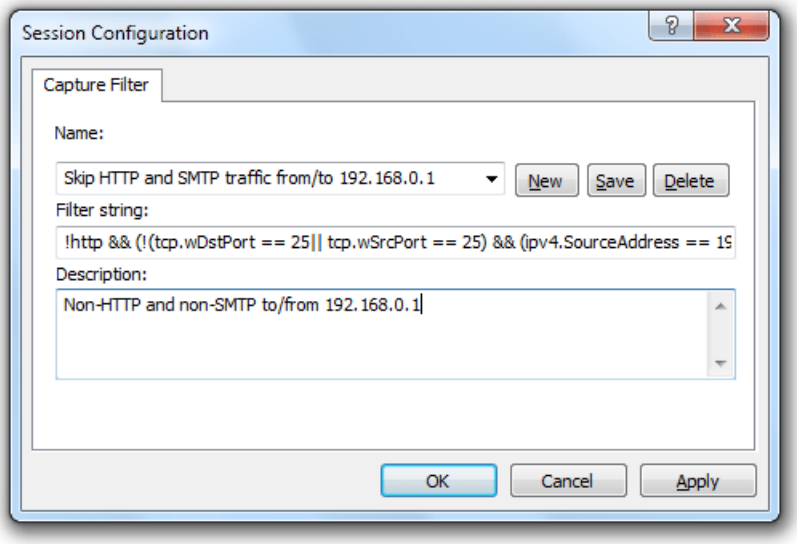
Filtering

Capture Filter

A monitoring session may have a special filter, called capture filter, configured. This filter is applied at the earliest point of time, before the monitored packet ever sent for data processing, including Data Recording. It allows the user to effectively filter unneeded packets out before sending them to expensive processing. Capture filter uses currently loaded protocol definitions.

Capture filter is specified in the Session Configuration Window.

You may select one of the pre-defined filters in the **Capture Filter** combobox, or press the **Edit** button to open the Capture Filter Configuration Window:



Use this window to select a display filter from a list, or enter the filter string manually, give it a name and save.

It is allowed to add, change and remove the capture filter of a running session.

Limitations

Serial Bridge and Remote data sources do not support capture filters.

Capture Filter Syntax

Capture filter expression is an expression that can reference fields of a bound protocol. It should evaluate to a `boolean` value. If the result of the expression is `true`, a packet is allowed to "pass", otherwise it is silently discarded. A result of the expression is automatically cast to `boolean` according to the following rules:

Expression Type	Conversion Rules
`boolean`	Used as is
`integer`	Zero is converted to `false`, any other value converted to `true`
`string`	Empty string is converted to `false`, any other string is converted to `true`
`Reference`	An invalid reference (a reference to non-existing field) is converted to `false`, otherwise it is `true`

Filter expression supports special kinds of immediates in addition to standard ones:

Immediate	Sample
IPv4 address	`127.0.0.1`
IPv6 address	`fe80::a4e0:281f:768b:ca30`
MAC address	`56:15:FB:B7:EF:99`

When the user types new filter expression, available fields are automatically suggested using the auto-completion engine. However, this engine is limited in its functionality and the user is advised to consult the source code of used protocols.

Examples

Serial Monitoring

The following filter passes only Serial Input/Output Control packets (IOCTL). This expression evaluates to `true` if and only if there is a sub-field `io` in the bound `serial` field:

```
C++
serial.io
```

Then following filter passes only Write packets:

```
C++
serial.Type == 4
```

The following filter passes only packets that has non-empty payload:

```
C++
(serial.Direction == "Up" && serial.Type == 3) || (serial.Direction == "Down" && serial.Type == 4)
```

USB Monitoring

The following filter passes only URB packets (discards PnP packets, for example):

```
C++
usb.urb
```

Network Monitoring

The following filter passes only IP traffic:

```
C++
ipv4 || ipv6
```

The following filter passes only packets sent to or received from `192.168.0.1`:

```C++
ipv4.SourceAddress == 192.168.0.1 || ipv4.DestinationAddress == 192.168.0.1
```

Generic Filtering (Legacy)

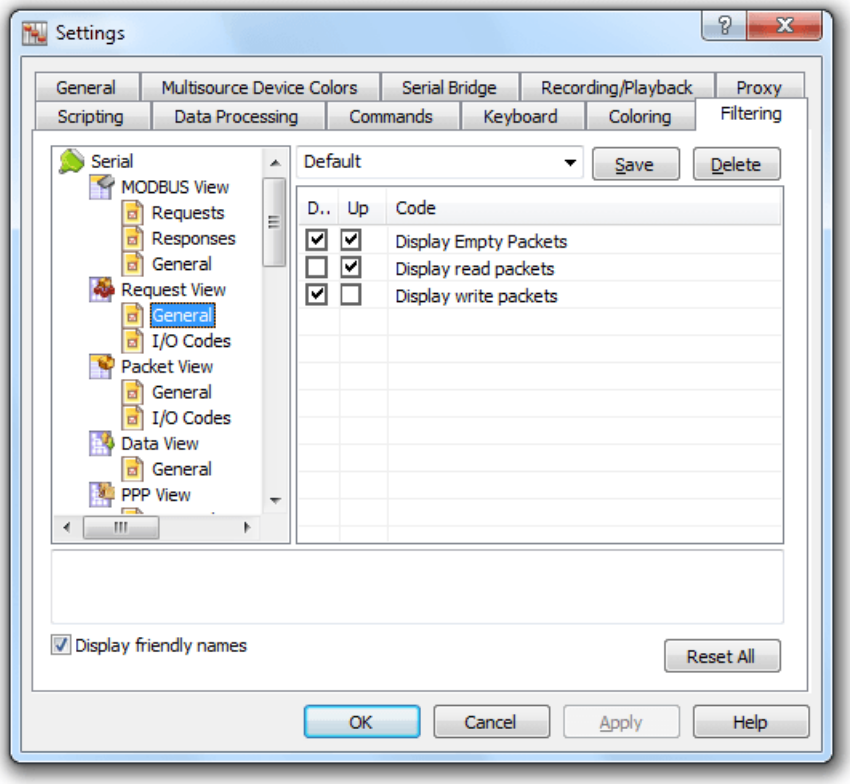
Device Monitoring Studio provides a generic mechanism that can be used to filter the output of different visualizers. You can configure several aspects of each supported visualizer.

You can configure filters and schemes in the Tools » Settings, Filtering Tab.

Each data visualizer that supports **Generic Filtering** checks each incoming packet if it suits the criteria specified by one or more filters. If packet fails to pass at least one filter rule, it is silently discarded.

Most data visualizers have non-empty default rule set that usually hides packets of lower interest, like answers for outgoing packets (that is, on their way **UP**).

Filtering Tab



The list of all filter groups is displayed on the left. Select a filter group to view its filters.

Click on the check box to enable or disable a particular filter. Some filter items have more than one check box, for example, you can configure whether the particular request is processed on its way **DOWN** or **UP**.

Click on the checkbox's header to enable/disable all checkboxes.

Working with Schemes

A scheme is a collection of specific filter settings. Each filter group has its own list of schemes. You can select the scheme from the combobox at the top of the page. Three built-in schemes appear for each

filter group:

Default

A default scheme sets each filter state to the default state. This scheme is selected by default for each filter group.

Display All

This scheme enables all filters - as a result, visualizers that use this filtering group will process all types of packets.

Display None

This scheme effectively turns off all filters.

You can create your own filtering schemes. To create a scheme, type the scheme name in the combobox and press the **Save** button. To delete a scheme, select a scheme from the list and press the **Delete** button. Note that you cannot delete a built-in scheme.

Per-Visualizer Scheme Application

In addition to system-wide filtering scheme selection, you can also apply a specific filtering scheme to a particular visualizer. To apply a filtering scheme, right-click on the visualizer window to bring up the popup menu (you should already have a running monitoring session), select **Filtering » Group Name » Filtering Scheme**. The selected filtering scheme will be applied to this visualizer only and will not affect any other opened visualizers as well as newly opened ones.

If you want to reset a filtering scheme to the one currently configured in the **Filtering Tab** for this visualizer, select the “Current” filtering scheme from the popup menu.

Advanced Features

Network Monitoring

Packet Builder

Network Monitor allows you to construct one or more packets and send them via the selected adapter to the network. New packets are created based on a template. **Network Monitor** comes with a set of predefined templates:

- ARP Packet
- IP Packet
- IPv6 Packet
- TCP Packet
- UDP Packet
- DHCP
- DHCP v6
- ICMP
- ICMP v6
- DNS
- LLMNR
- NbtNs
- NbtSs

Template is an empty packet of some type, that is a packet, which does not have any payload, but does have the infrastructure: flags, sizes and selectors.

A template may be used as a starting point to construct more complex packet. For example, you may start with an IP packet template and eventually construct an HTTP packet. You may also use this technique to construct packets for user-defined protocols (if they are loaded into **Network Monitor**).

A constructed packet may be saved as a template using the **Packet Builder » Save as Template...** command.

Packet Editing

After the packet is added to the list, you may edit it either using the binary representation (lower pane) or its decoded form (upper pane). The binary pane also supports copying and pasting, so you can copy a monitored packet from one of data visualizers and paste it into the binary pane to get a copy of the packet. This copy may later be edited and sent back to the network.

Both panes are synchronized: you can edit the same packet in one or both panes.

> **NOTE**
>
> Some protocols have a checksum field that is used to check if the packet is valid or not. **Network Monitor** provides automatic checksum calculation for IP and TCP packets. You must manually compute and update checksum for any other kind of packet that have checksum incorporated. Other protocols also have payload size. **Network Monitor** does not automatically update any size field!

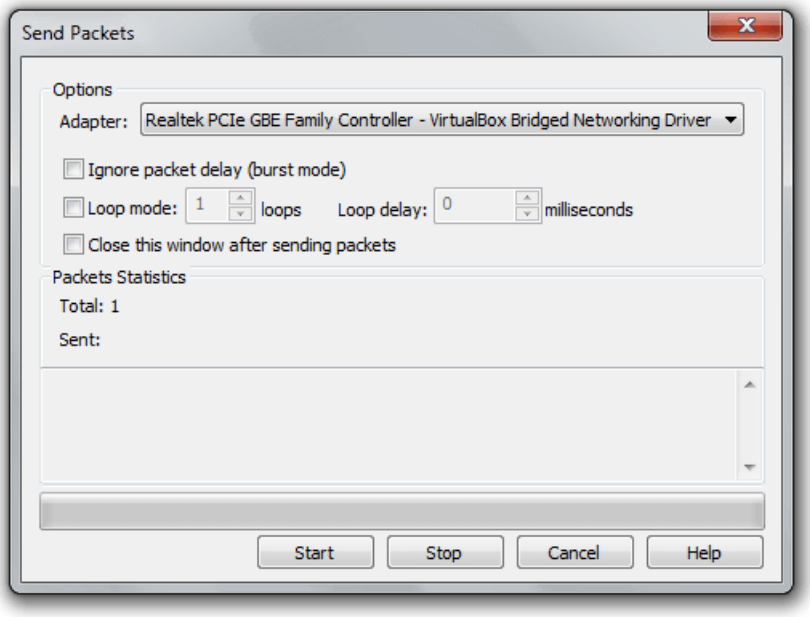
The upper pane displays the packet contents according to the loaded protocol definition files. Click on the small plus sign to expand a sub-tree and click on the small minus sign to collapse the sub-tree. Double-click the line to edit the value. After you finish entering the new value, click the Enter key to save changes or Esc key to discard them.

You are restricted to editing only the "leaf" values (fields that cannot be further expanded). In addition, **Network Monitor** has built-in parsers for the following network addresses:

- MAC address
- IPv4 address
- IPv6 address

For these fields, you may double-click the address field and enter the new value directly (like `127.0.0.1`). Your input will be parsed accordingly.

Sending Packets



Use either the **Packet Builder » Send Packet...** command or **Packet Builder » Send All Packets...** command to send the prepared packets to the network.

First, select the adapter you would like to use for sending. Other options let you ignore the specified packet delay, enable loop mode and specify whether you want to automatically close the **Send Packets** window after all packets are sent.

To start sending, press the **Start** button. To stop sending packets, press the **Stop** button.

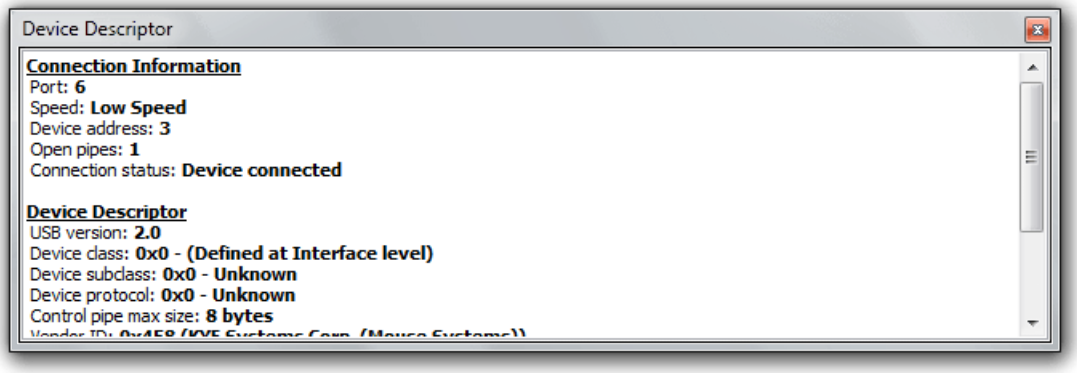
Saving and Loading Packets

Network Monitor allows you to save the prepared packet to a file (in the binary form). You may later load this packet from a file. This feature also allows you to treat any binary file as contents of a packet.

In addition to binary export, you may copy the decoded value or the entire line to the Clipboard by bringing up the context menu and selecting either the **Copy Value** or **Copy Line** commands.

USB Monitoring

Device Descriptor



This tool window displays the decoded USB device descriptor. You can scroll the contents of the window to view the whole information if it is larger than window. Use the keyboard or mouse to select the text in the window and select the **Edit » Copy** command to copy the selected text into the Clipboard.

Displayed Information

This window displays the following information:

Connection Information

Port: **hub port number**
Speed: **device speed (Low Speed, Full Speed, High Speed or Super Speed)**
Device address: **device bus address**
Open pipes: **the number of open pipes**
Connection status: **the status of the connection**

Device Descriptor

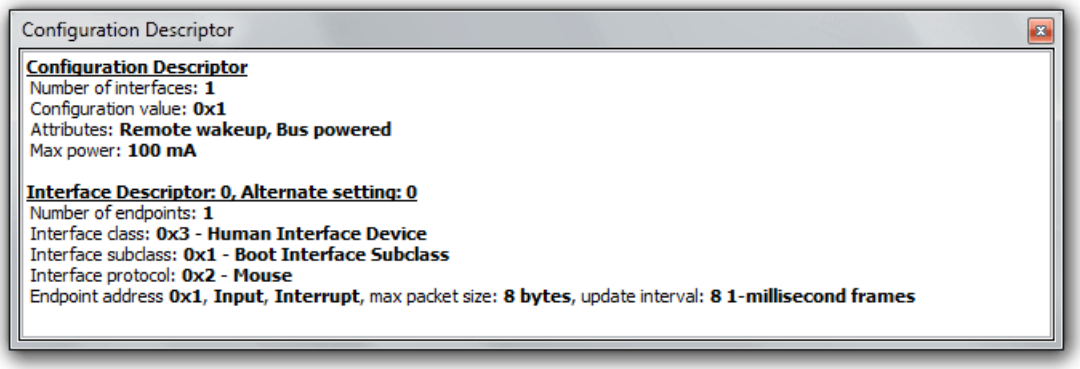
USB Version: **USB compatible version (1.1, 2.0 or 3.0)**
Device class: **device class code and description**
Device subclass: **device subclass code and description**
Device protocol: **device protocol code and description**
Control pipe max size: **the maximum size of the control pipe transfer**
Vendor ID: **USB consortium-assigned vendor identifier**
Product ID: **vendor-assigned product ID**
Product version: **vendor-assigned product version**
Manufacturer: **manufacturer name, taken from string descriptor**
Product: **product name, taken from string descriptor**
Serial Number: **product serial number, taken from string descriptor**
Configurations: **the number of supported configuration**

Parsing Identifiers

The Device Monitoring Studio uses the `usb.ids` file, kindly provided by Vojtech Pavlik vojtech@suse.cz and Stephen J. Gowdy gowdy@slac.stanford.edu, to properly parse numeric identifiers into human-readable string values.

The file is installed into the product installation folder and is automatically scanned on startup. The mentioned file is frequently updated on-line. Please visit the link from time to time to download the most recent version.

Configuration Descriptor



This tool window displays the decoded USB configuration descriptor, which consists of configuration, interface and endpoint descriptors. You can scroll the contents of the window to view the whole information if it is larger than window. Use the keyboard or mouse to select the text in the window and select the **Edit » Copy** command to copy the selected text into the Clipboard.

Displayed Information

This window displays the following information:

Configuration Descriptor

Number of interfaces: **total number of supported interfaces**
Configuration value: **configuration ordinal number**
Attributes: **device attributes, separated by comma**
Max power: **maximum power consumption, in mA**

Interface Descriptor: N, Alternate Setting: M

Number of endpoints: **the number of endpoints (pipes)**
Interface class: **interface class code and description**
Interface subclass: **interface subclass code and description**
Interface protocol: **interface protocol code and description**
Endpoint address X, Input (Output), Type (Bulk, Interrupt or Isochronous), max packet size: Z, poll period - **for each endpoint**

Parsing Identifiers

The Device Monitoring Studio uses the `usb.ids` file, kindly provided by Vojtech Pavlik vojtech@suse.cz and Stephen J. Gowdy gowdy@slac.stanford.edu, to properly parse numeric identifiers into human-readable string values.

The file is installed into the product installation folder and is automatically scanned on startup. The mentioned file is frequently updated on-line. Please visit the link from time to time to download the most recent version.

Dependent Devices

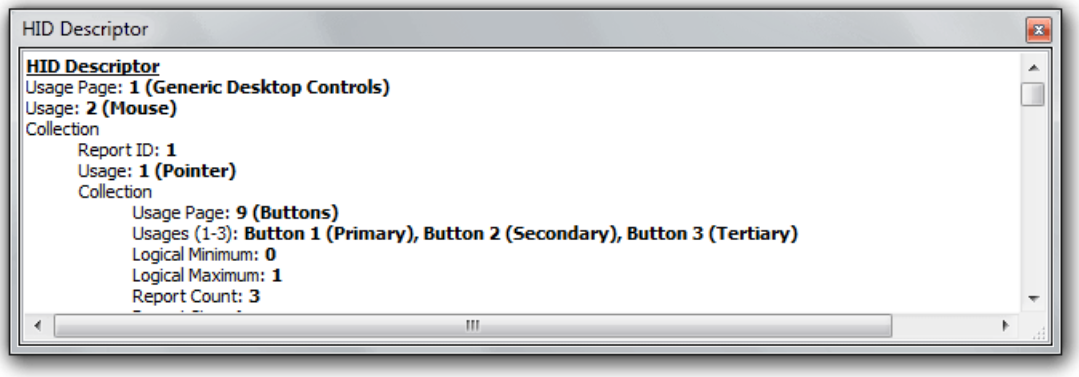
This tool window displays the tree of devices created by operating system for a device selected in Devices Tool Window. At the tree root, the selected USB device is displayed. If an operating system creates one or more other devices, they are displayed as children of the selected device.

For each device in the tree, the following three options are available for you: you can restart the device, view its properties through Device Manager or copy the full device name into the Clipboard.

NOTE

Please note that operating system can block the restart request if any other program actively uses the selected device.

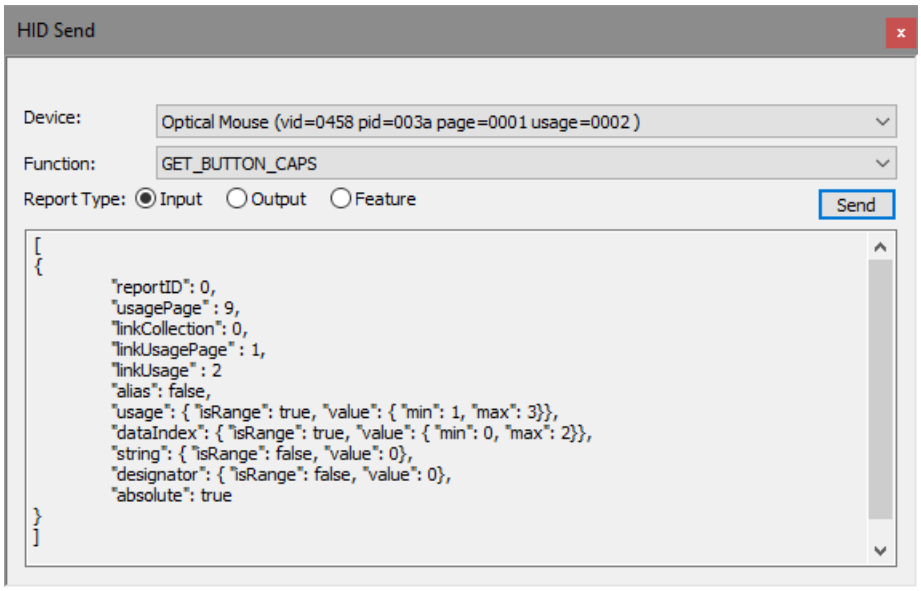
HID Descriptor



This tool window displays the decoded USB HID descriptor, available for devices belonging to USB HID class. This window displays HID Report descriptors. You can scroll the contents of the window to view the whole information if it is larger than window. Use the keyboard or mouse to select the text in the window and select the **Edit » Copy** command to copy the selected text into the Clipboard.

HID Send

HID Send module allows the user to directly communicate with HID devices. The user may query HID device parameters, construct and send HID reports.



First, select the device from the **Device** drop-down. Then select a function from the **Function** drop-down. Select the report type and provide any additional parameters (parameters will automatically appear when corresponding function is selected).

Several functions require the report bytes to be specified.

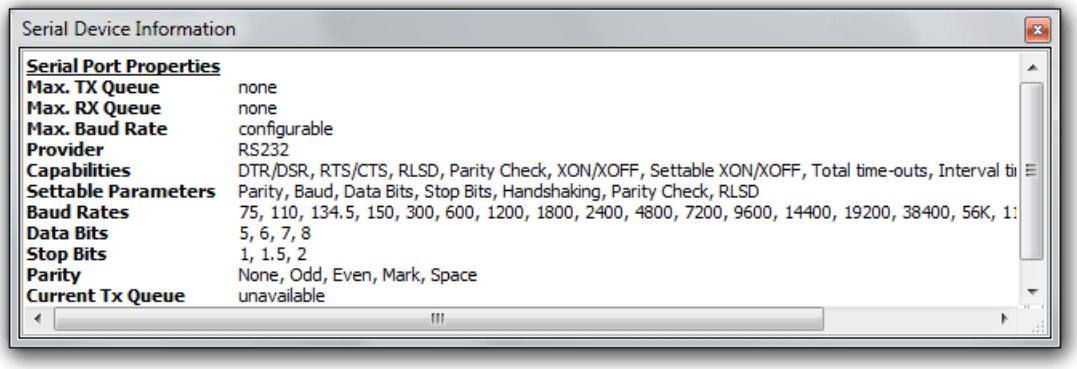
When you finish configuring the request, click the **Send** button. You will see the operation result in the result pane.

Scripting Support

HID Send module may be fully controlled with scripting. Please refer to the documentation of HID Manager Object for more information.

Serial Monitoring

Serial Device Information



This tool window displays the information for the selected serial device. You can scroll the contents of the window to view the whole information if it is larger than window. Use the keyboard or mouse to select the text in the window and select the **Edit » Copy** command to copy the selected text into the Clipboard.

Displayed Information

This window displays the following information:

Max. TX Queue

Specifies the maximum size, in bytes, of the driver's internal output buffer. A value of “none” indicates that the serial provider imposes no maximum value.

Max. RX Queue

Specifies the maximum size, in bytes, of the driver's internal input buffer. A value of “none” indicates that the serial provider imposes no maximum value.

Max. Baud Rate

Specifies the maximum allowable baud rate, in bits per second (bps). A value of “configurable” means the user can set any baud rate.

Provider

Specifies the specific communications provider type.

Capabilities

Specifies a list of the capabilities offered by the provider.

Settable Parameters

Specifies a list of the communications parameters that can be changed.

Baud Rates

Specifies a list of the baud rates that can be used.

Data Bits

Specifies a list of the number of data bits that can be set.

Stop Bits

Specifies a list of the stop bit that can be selected.

Parity

Specifies a list of the parity settings that can be selected.

Current Tx Queue

Specifies the size, in bytes, of the driver's internal output buffer. A value of "unavailable" indicates that the value is unavailable.

Current Rx Queue

Specifies the size, in bytes, of the driver's internal input buffer. A value of unavailable indicates that the value is unavailable.

Compatibility Notes

Although the device may seem to report specific combination, this is not always the case. For example, if the device reports supporting data bits as "5, 6, 7, 8" and stop bits as "1, 1.5, 2 stop bits" it does not mean that it will support some specific combination, such as "8 bit 2 stop bits". There is no way in Windows to determine whether the specific combination is supported by hardware.

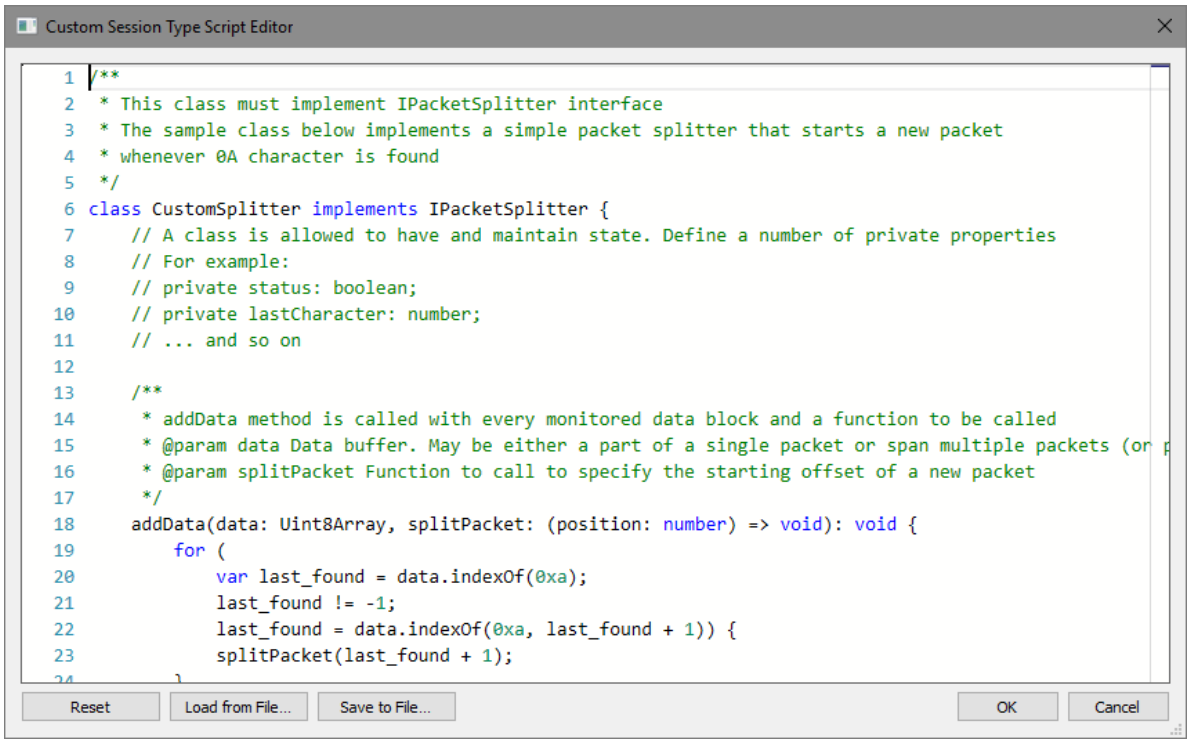
Although the serial device may seem to report the specific settings combination and even may actually support it, it does not mean that the device connected to this serial port will support it as well.

Custom Communication Mode

In addition to a number of built-in session types supported by Device Monitoring Studio for serial monitoring, the user may also define custom rules for packet joining and splitting.

This is done by means of TypeScript (or JavaScript) custom code. This topic describes the specifications the custom script must adhere to. First, in order to start editing a custom script, the user must select the "Custom" session type in corresponding dialog and then press the **Edit** button.

The following window appears:



Use this window to edit a custom script. A window automatically highlights syntax errors. Use your

mouse to hover over a highlighted error to see explanation.

If you have a custom script stored in a separate file, click the **Load from File** button to load it. If you need to store the custom script outside of Device Monitoring Studio, click the **Save to File** button. To reset to default sample implementation, click the **Reset** button.

When you open this window for the first time, a sample script is automatically loaded. It contains an implementation of a simple packet splitter that ends a packet whenever a character `0A` is received. The code contains enough comments and should be self-explanatory.

Custom Splitter Code Structure

Serial source creates two isolated execution environments upon session start. One execution environment is used to process read requests (received data) and another is used to process write requests (sent data). These environments are isolated from each other and from any other scripts running in Device Monitoring Studio. That means that they cannot share any variables and call functions from different environments.

After an execution environment is created, a function called `createSplitter` is called. It must have the following signature:

```TypeScript
function createSplitter(type: PacketType): IPacketSplitter;
```

Where `PacketType` is defined as:

```TypeScript
enum PacketType {
  Read,
  Write
}
```

and `IPacketSplitter` as:

```TypeScript
interface IPacketSplitter {
  addData(data: Uint8Array, splitPacket: (position: number) => void): void;
}
```

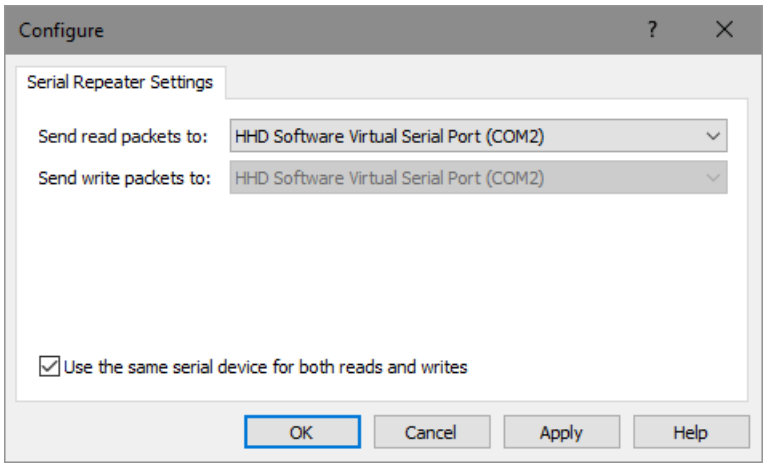
`createSplitter` function is passed a splitter type (either `PacketType.Read` or `PacketType.Write`) and must create an instance of a class that implements the `IPacketSplitter` interface.

For each captured data block, `IPacketSplitter.addData` method is called. It is passed a copy of data block and a function to be called when custom code decides there is a start of a new packet in this data. `splitPacket` function must be called with an offset within the passed data block. It must be between `0` and data block size (inclusive).

Please refer to the sample code for more details.

Data Repeater

Data Repeater is a special data processing module, which may be added to a serial or serial playback monitoring session. When added, it requires the user to provide a port where read packets should be sent and a port where write packets should be sent. The user may use the same serial device for both packet types.



When the monitoring session is started, all captured data is redirected (repeated) to given serial device(s).

Serial Terminal

Serial Terminal module implements the serial terminal emulation, which is built into the Device Monitoring Studio user interface.

You can configure as many serial terminal sessions as you need, with a maximum of one session for every serial device installed on your computer. The Serial Terminal supports various serial port settings, including baud rate, parity, flow control, etc.

Create a new terminal session using the **Tools » Serial Terminal » New Terminal Session** command. The Serial Terminal Session Configuration Window is displayed to let you configure new session. After the session is created, it is added as an entry in the **Serial Terminal** root menu. The session window is opened in client area.

In addition, you may right-click the serial device in Devices Tool Window and select the **Start Terminal...** command.

Immediately after the session is created, the Serial Terminal starts listening for incoming data from the serial device. All received data are displayed in the terminal window.

To send data to the serial device, start typing it on the keyboard. The echo option, which is controlled by the corresponding option in a context menu, governs the echoing of the sent characters. You may find it useful in some situations. It is recommended, although, to turn off the option if the device you are communicating with provides its own echoing.

You can also send the contents of a text files to the serial device by using the **Send File...** command in the context menu. You can select one of the two send methods. First of them will split file contents to multiple lines and send them to device sequentially (with `0x0D` symbol in the end of each line). Second method will send the file contents without splitting (but using fixed size buffers). To select the method please go to Tools » Settings, General Tab and check the “Send text lines to file one by one” option.

The data from file is sent as sequential blocks of some finite size. You can specify the block size with the Block size option (“Serial Terminal” group in Tools » Settings, General Tab). The minimum value is 1 and maximum is 65535 bytes. The default block size is 1024 bytes.

Use the **Advanced Send** option in context menu to enter the pattern to send. In addition to pattern, you may specify delays and looping conditions.

Integration with Serial Monitoring

The Serial Terminal module works independently from the serial monitoring module, allowing you to see the "low-level" of established serial terminal session. It may be useful to use both modules at the same time for a single serial device to solve a number of tasks, including the following:

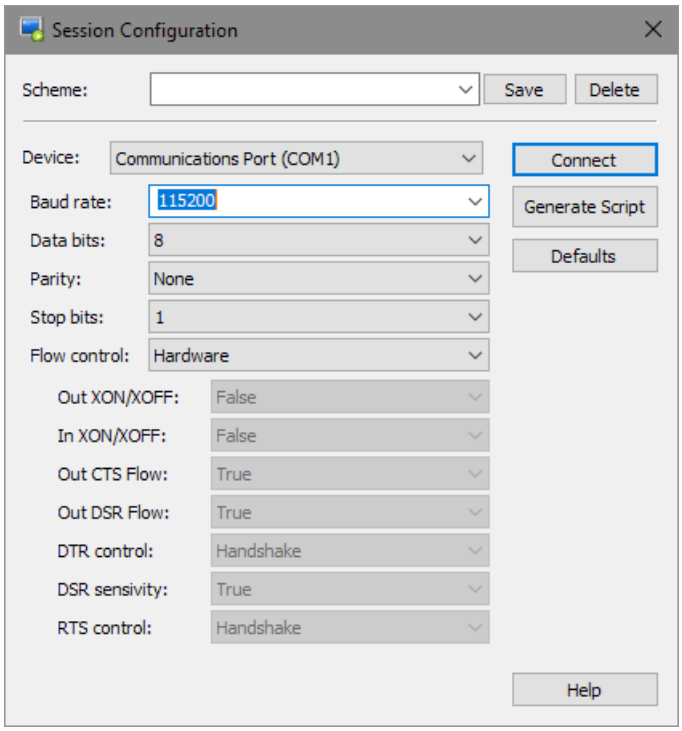
- Debugging the serial interface of a hardware device attached to the serial port.
- Decoding the binary response to a text request.
- Working in conjunction with a user script running in the Device Monitoring Studio's environment. See the section below for more information on scripting.
- Starting with version 8, the integration is even more tight - when you configure a serial monitoring session, you have an option to automatically launch terminal session with hidden or visible window.

Scripting Support

The Serial Terminal module exposes the fully scriptable object to be used by scripts running in the Device Monitoring Studio environment. Please consult to the Scripting section for more information on built-in scripting and to the Serial Terminal Object section for description of the interfaces, exposed by the Serial Terminal module.

Session Configuration Window

This dialog allows you to configure new serial terminal session.



Select the serial device from the list of detected serial devices, specify connection baud rate, parity, byte size and a number of stop bits, as well as flow control operation and press the **Connect** button.

Optionally enter the scheme name and click the **Save** button to save the session configuration for later use. The dialog also stores the last settings and displays them next time you create a serial terminal session.

Generating Script

Click the **Generate script** button to generate script that creates a new terminal session and sets its parameters according to settings in the configuration window.

MODBUS Send

MODBUS Send window provides an easy way to control a MODBUS-compatible device. It can come of great help when you need to debug your device, for example verify its responses. You can set all parameters for any standard or user-defined MODBUS function visually and then just click on the Send button. Result rollout will dynamically reflect the changes you make.

MODBUS Send window allows you to send both requests and responses.

It is convenient to use a MODBUS Viewvisualizer in conjunction with a MODBUS Send window for two-way debugging (to view both requests and responses).

Using MODBUS Send

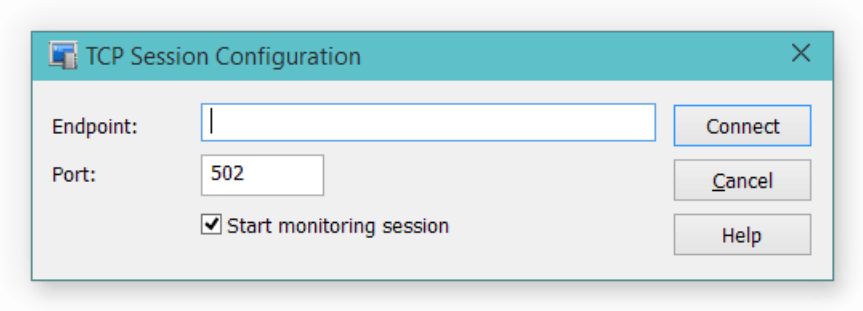
MODBUS Send with Serial Devices

Below are the generic steps you need to perform to send a MODBUS request to the compatible device:

1. Create a terminal session and then switch to MODBUS Send window (use the **View » Tool Windows » MODBUS Send** command if it's not visible). Alternatively, click the **New** button in MODBUS Send window and select "Serial Session" option.
2. Choose the correct terminal session from the list of terminal sessions in MODBUS Send window. If there is only one active terminal session, it will be chosen automatically.
3. Choose the correct mode (ASCII or RTU) from the list of modes in MODBUS Send window. Please refer to MODBUS documentation for more information.
4. Select the correct MODBUS device address (0 is used for broadcasting). Please refer to MODBUS documentation for more information.
5. Select the function you need from the Function combo box. MODBUS Send window will update itself according to the function selected. This may cause new rollouts to appear and some to disappear.
6. Open the function parameter rollout. It can be the generic Parameters rollout, or function-specific rollout. See the detailed function description below for information on particular function and its configuration rollouts.
7. Enter all required data in the function configuration rollout.
8. You can close the rollout - all entered data will be saved. If you re-open the rollout, you will see all entered data intact.
9. You can examine the resulting packet contents in the Result rollout.
10. Press the **Send** button to send data to the device.

MODBUS Send with TCP Session (MODBUS TCP Protocol)

Create a TCP session by pressing the **New** button and selecting "TCP Session" option from the list. Configure the TCP session by filling the destination address and port fields. Optionally enable the "Create monitoring session" option to automatically start a network monitoring session for this endpoint:



After the TCP session is created, continue with a normal work flow as described in the previous section.

MODBUS Session

MODBUS Send with Serial Devices

MODBUS Send window usesTerminal Session to connect to the serial device. A terminal session can be created in Terminal tool window or by the running script. It will then appear immediately in the **Session list** in the MODBUS Sendwindow. When the terminal session is closed, it is automatically removed from the list.

You cannot use MODBUS Send window if you have no serial devices installed on your computer.

The MODBUS Send window controls stay disabled until you configure and select a terminal session.

MODBUS Send over MODBUS TCP Protocol

MODBUS Send window usesTCP Session to connect to the device over IP network. A TCP session can be created by pressing the New button on MODBUS Send window or by the running script. It will then appear immediately in the Session list in the MODBUS Send window. When the TCP session is closed, it is automatically removed from the list.

You cannot use MODBUS Send window if you have no configured TCP sessions.

The MODBUS Send window controls stay disabled until you configure and select a TCP session.

MODBUS Send Window Rollouts

Generic Rollouts

Result Rollout

Result rollout is intended to help you analyze the packet before you send it. The rollout is hidden by default, but you can open it by clicking on its name. Result rollout includes read-only text control and Total Length control. When you modify MODBUS function parameters, Result rollout automatically reflects the changes. It is very helpful for MODBUS protocol exploration.

The values are displayed in Result rollout in hex format grouped as single bytes. For example, the 255 value is represented as FF.

Note that the Result rollout displays empty data if invalid or incomplete values are specified for one of the function parameters.

As long as MODBUS packet is limited in size, the Result rollout displays maximum 252 bytes of data (as defined in MODBUS protocol documentation). First special leading byte is not displayed in Result rollout, so the maximum size of data in this case is 253.

> **NOTE**
>
> The result is displayed in binary form (it is not converted to ASCII) without leading ":" character in ASCII mode.

The context menu offers you the following functions:

Save to binary file

Binary file is a non-text file with a bin extension. The data is written in the binary format. For example, the `DD AA B5` (3 bytes) sequence displayed in result pane will result in bytes `0xDD`, `0xAA` and `0xB5` written into the output file. The resulting file can subsequently be used by some other application for additional processing.

Save to file

Saves the contents of the Result rollout to the ASCII-encoded text file. The file then can be opened in any text editor or used by some other application.

Copy

Copies the contents of the Result rollout to the Clipboard in text format.

User Data Rollout

The User Data rollout appears when the USER-DEFINED FUNCTION is chosen from the Function list. It allows you to enter the contents of the custom request. Type in hexadecimal numbers (characters 0-9, A-F, either lowercase or uppercase).

The context menu offers you the following functions:

Save to binary file

Binary file is a non-text file with a bin extension. The data is written in the binary format. For example, the `DD AA B5` (3 bytes) sequence displayed in result pane will result in bytes `0xDD`, `0xAA` and `0xB5` written into the output file. The resulting file can subsequently be used by some other application for additional processing.

Save to file

Saves the contents of the Result rollout to the ASCII-encoded text file. The file then can be opened in any text editor or used by some other application.

Load from file

Loads the data from either the text or binary file, which was previously saved with a **Save to binary** or **Save to file** commands.

Clear

Empties the entered data.

Parameters Rollout

This rollout is used in the following functions:

- Read Coil Status
- Read Discrete Inputs
- Read Holding Registers
- Write Single Coil
- Write Single Register
- Diagnostics

- Read FIFO Queue
- Write Single Coils (response)
- Write Single Register (response)
- Read Exception Status (response)
- Diagnostics (response)
- Get Comm Event Counter (response)
- Write Multiple Coils (response)
- Write Multiple Registers (response)
- ERROR (response)

The rollout contains two edit boxes that allow you to enter two 16-bit integers (except ERROR response, where only 8-bit values are allowed). Since the Read FIFO Queue function has a single parameter, the second edit box is disabled when you select this function.

Parameter names varies according to the selected function. Please consult the MODBUS documentation for a definition of function parameters and their acceptable ranges.

Request Rollouts

Write Multiple Coils Rollout

This rollout is used to configure Write Multiple Coils function parameters. It consists of Starting Address field, Number of Coils field and a list of coils. Coil is a boolean value.

Usage

Below are the steps you need to perform to configure the Write Multiple Coils function parameters properly.

1. Enter the starting address value in the Starting Address edit box. Starting Address is a value between 0 and 65535.
2. Enter the number of coils into the Number Of Coils edit box. See the items in the list added or removed.
3. Check the items (coils) you want to set into TRUE state. Un-check items to set them into the FALSE state.

The context menu offers you the following functions:

Check All

Puts all added coils to the TRUE state.

Uncheck All

Puts all added coils to the FALSE state.

Invert All

Inverts the state of all coils.

Write Multiple Registers Rollout

This rollout is used to configure Write Multiple Registers function parameters. It consists of Starting Address field, Number of Registers field and a list of registers. A register is a 16-bit value.

Below are the steps you need to perform to configure the Write Multiple Registers function parameters properly.

1. Enter the starting address value in the Starting Address edit box. Starting Address is a number from

0 to 65535.
2. Enter the number of registers in the Number of Registers edit box. It can be a number from 0 to 120. See the registers added to the list or removed from it. By default, all of them have a zero value.
3. To change the value of register, select it in the list, then enter the register value into the Register Value edit box and click the Set button (or press the ENTER key).
4. Repeat the previous step for each register.

Note: you can use “fast register entering” method. It is used when the number of registers is too big. Repeat these steps:

1. Select the first register in list.
2. Enter register value into the Register Value edit box.
3. Press the ENTER key. The highlighted register's value will be updated and the next register becomes active. Repeat steps 2 and 3 for each remaining register.

The context menu offers you the following functions:

Save to file

Saves all entered register values to comma-separated file (.CSV).

Load from file

Loads register values from comma-separated file (.CSV).

Read File Record Rollout

This rollout is used to configure Read File Record function parameters. It consists of File Number, Record Number, Register Length edit boxes and a list. Each file record you enter should have its own file number value, record number value and register length value.

Usage

Below are the steps you need to perform to configure the Read File Record function parameters properly.

1. Enter file number into the File Number edit box. Valid numbers are from 0 to 65535.
2. Enter record length into the Record Length edit box. Valid numbers are from 0 to 65535.
3. Enter register length into the Register Length edit box. Valid numbers are from 0 to 65535.
4. Click the Add button to add a file record to the list.
5. Repeat steps 1-4 as many times as you need.

Use the && button to remove currently selected file record from the list. Use the **Remove All** button to remove all file records from the list.

The context menu offers you the following functions:

Remove

Removes the currently selected file record from the list.

Remove All

Remove all file records from the list.

Write File Record Rollout

This rollout is used to configure Write File Record function parameters. It consists of File Number, Record Number, Register Length edit boxes and a list. Each file record you enter should have its own file number value, record number value and register length value.

Usage

Below are the steps you need to perform to configure the Write File Record function parameters properly.

1. Enter file number into the File Number edit box. Valid numbers are from 0 to 65535.
2. Enter record length into the Record Length edit box. Valid numbers are from 0 to 65535.
3. Enter register length into the Register Length edit box. Valid numbers are from 0 to 65535.
4. Click the Add button to add a file record to the list.
5. Enter data in the Edit Data dialog. See User Data Rollout section for more information.
6. Repeat steps 1-5 as many times as you need.

Use the **Remove** button to remove currently selected file record from the list. Use the **Remove All** button to remove all file records from the list.

The context menu offers you the following functions:

Remove

Removes the currently selected file record from the list.

Remove All

Remove all file records from the list.

Show Data

Brings up the Edit Data dialog in read-only mode.

Mask Write Register Rollout

This rollout is used to configure Mask Write Register function parameters. It consists of Reference Address edit box, sixteen OR Mask check boxes and sixteen AND Mask check boxes.

Usage

Below are the steps you need to perform to configure the Mask Write Register function parameters properly.

1. Enter reference address value into 'Reference Address' edit box. It can hold values from 0 to 65535.
2. Check the check boxes in OR Mask group. Each check box represents single bit in the mask. The mask in the Result box is dynamically updated as you check or un-check bits.
3. Check the check boxes in AND Mask group. Each check box represents single bit in the mask. The mask in the Result box is dynamically updated as you check or un-check bits.

Read/Write Multiple Registers Rollout

This rollout is used to configure Read/Write Multiple Registers function parameters. It consists of Read Address, Read Quantity, Write Address, Write Quantity fields and a list. Register is a 16-bit value.

Usage

Below are the steps you need to perform to configure the Mask Write Register function parameters properly.

1. Enter the starting read address in the Read Address edit box. Valid values are from 0 to 65535.
2. Enter the number of registers in the Read Quantity edit box. Valid values are from 0 to 118.
3. Enter the starting write address in the Write Address edit box. Valid values are from 0 to 65535.

4. Enter the register number in the Write Quantity edit box. See the registers added to the list or removed from it. By default, all of them have a zero value.
5. To change the register value, select it, then enter the value into the Register Value edit box and press the Set button (or press the ENTER key).
6. Repeat the previous step until all registers are added.

Note: you can use “fast register entering” method. It is used when the number of registers is too big. Repeat these steps:

1. Select the first register in list.
2. Enter register value into the Register Value edit box.
3. Press the ENTER key. The highlighted register's value will be updated and the next register becomes active. Repeat steps 2 and 3 for each remaining register.

The context menu offers you the following functions:

Save to file

Saves all entered register values to comma-separated file (.CSV).

Load from file

Loads register values from comma-separated file (.CSV).

Response Rollouts

Get Comm Event Log Rollout (response)

This rollout is used to configure Get Comm Event Log function response parameters. It consists of Status field, Message Count field, Event Count field and list of events.

Below are the steps you need to perform to configure the Get Comm Event Log function response parameters properly.

1. Enter status value into Status field. Status is a number from 0 to 65535.
2. Enter message count value into Message Count field. It can be a number from 0 to 65535.
3. Enter event count value into Event Count field. It can be a number from 0 to 245.
4. Click on any other control to confirm the event count value. See the events added into list or removed from it. By default, all of them have 0 value.
5. To change the value of event, do the following: select the event in a list, enter event value into Event Value field, click the Set button (or press ENTER).
6. Repeat previous step several times until you've finished.

Note: you can use “fast register entering” method. It is used when there is a large number of registers. Repeat these steps:

1. Select the first event in a list.
2. Enter event value into the Event Value field.
3. Press the ENTER key. The highlighted event value will be updated and the next register becomes active. Repeat steps 2 and 3 for each remaining register.

The context menu offers you the following functions:

Save to file

Saves all entered event values to comma-separated file (.CSV).

Load from file

Loads event values from comma-separated file (.CSV).

Read FIFO Queue Rollout (response)

This rollout is used to configure Read FIFO Queue function response parameters.

Below are the steps you need to perform to configure the Read FIFO Queue function response parameters properly.

1. Enter the number of FIFO registers value in FIFO count field. It can be a number from 0 to 120. See the registers added into the list or removed from it. By default, all of them have 0 value.
2. To change the value of register, do the following: select the register in the list, enter register value into FIFO Value field, click the Set button (or press ENTER)
3. Repeat previous step several times until you've finished.

Note: you can use “fast register entering” method. It is used when there is a large number of registers. Repeat these steps:

1. Select the first register in list.
2. Enter value into the FIFO Value field.
3. Press the ENTER key. The highlighted FIFO register value will be updated and the next item becomes active. Repeat steps 2 and 3 for each remaining register.

The context menu offers you the following functions:

Save to file

Saves all entered register values to comma-separated file (.CSV).

Load from file

Loads register values from comma-separated file (.CSV).

Read File Record Rollout (response)

This rollout is used to configure Read File Record function response parameters. It consists only of a list.

Below are the steps you need to perform to configure the Read File Record function response parameters properly.

1. Press the **Add** button. The data-entering dialog box will appear.
2. Enter the data and press **OK** button. Record will be added to the list.
3. Repeat steps 1-2 until you've finished.

Use the **Remove** button to remove currently selected file record from the list. Use the **Remove All** button to remove all file records from the list.

The context menu offers you the following functions:

Remove

Click this command to remove the currently selected file record from the list.

Remove All

Click this command to remove all file records from the list.

Read/Write Multiple Registers Rollout (response)

This rollout is used to configure Read-Write Multiple Registers function response parameters.

Below are the steps you need to perform to configure the Read-Write Multiple Registers function response parameters properly.

1. Enter the number of registers value in Register Count field. It can be a number from 0 to 120. See the registers added into the list or removed from it. By default, all of them have 0 value.
2. To change the value of register, do the following: select the register in the list, enter register value into Register Value field, click the Set button (or press ENTER).
3. Repeat previous step until you've finished.

Note: you can use “fast register entering” method. It is used when there is a large number of registers. Repeat these steps:

1. Select the first register in list.
2. Enter register value into the Register Value edit box.
3. Press the ENTER key. The highlighted register's value will be updated and the next register becomes active. Repeat steps 2 and 3 for each remaining register.

The context menu offers you the following functions:

Save to file

Saves all entered register values to comma-separated file (.CSV).

Load from file

Loads register values from comma-separated file (.CSV).

Report Slave ID Rollout (response)

This rollout is used to configure Report Slave ID function response parameters. It consists of Run Indicator Status field, Slave ID field, Additional Data field. The Slave ID and Additional Data are variable length fields (device specific). Below are the steps you need to perform to configure the Get Comm Event Log function response parameters properly.

1. Enter run indicator status value into Run Indicator Status field. It can be a number from 0 to 255. Enter 0 (0x00) for FALSE and 255 (0xFF) for TRUE.
2. Enter slave ID data (variable length data) into the Slave ID field. You can enter maximum 16 bytes in this field.
3. Enter additional data (variable length data) into the Additional Data edit box. You can enter maximum 230 bytes in this field.

Protocols

Device Monitoring Studio supports automatic parsing of monitored USB, Serial and Network data according to custom protocol. The protocol definition language is a C-like programming language with full support of dynamic-size data structures. It supports conditional and loop statements, has a rich set of built-in functions, allows user-defined functions and supports extension via external JavaScript code.

Protocol Binding Workflow

Device Monitoring Studio automatically binds monitored Serial, USB and Network packets if aCapture Filter is enabled for a session or at least one protocol-based data processing module is added to it. Examples of protocol-based modules include: Structure View, Custom View, Request View, Text Exporter and others.

After the packet binding is complete, the filter is applied (if enabled). At this stage, a packet may be discarded. Note that this occurs before any other component receives a bound packet. After that, field values become available for all data processing modules.

Structure View is a data visualizer that displays all bound fields as they are. Thus it automatically works with any built-in or custom protocol. **Text Exporter** is an “exporting” version of **Structure View**: it

directs all parsed data into the text file instead of a window on a screen.

Custom View data visualizer may be used to get bound packet field values, format them and display in a visualizer window. Serial Monitor's Request View and Console View data visualizers as well as Network Monitor's HTTP View are controlled by Custom View.

Pre-installed Protocols

Network Monitoring Protocols

Network Monitor comes with many pre-installed protocols. All supported protocols, along with a short description and file name where they are defined, are listed in the table below:

Protocol	Description	Protocol Definition File
`Arp`	Address Resolution Protocol	`arp.h`
`Bvlc`	BACnet Virtual Link Control Protocol	`bvlc.h`
`Bacnet`	BACnet Protocol	`bvlc.h`
`BacnetMSTP`	BACnet MSTP (Master Slave Token Passing) Protocol	`bvlc.h`
`Ccp`	CAN Calibration Protocol Protocol	`ccp.h`
`Chap`	Challenge Handshake Authentication Protocol	`chap.h`
`Dhcp`	Dynamic Host Configuration Protocol (over Ipv4)	`dhcp.h`
`Dhcpv6`	Dynamic Host Configuration Protocol (over Ipv6)	`dhcpv6.h`
`Dns`	Domain Name System Protocol	`dns.h`
`Ethernet`	Ethernet Protocol	`net_defs.h`
`Gre`	Generic Routing Encapsulation Protocol	`gre.h`
`HTTP`	Hyper-text Transfer Protocol	`http.h`
`llMnr`	Link Local Multicast Name Resolution	`dns.h`
`Icmp`	Internet Control Message Protocol (over Ipv6)	`icmp.h`
`Icmpv6`	Internet Control Message Protocol (over Ipv6)	`icmpv6.h`
`LLDP`	Link Layer Discovery Protocol	`lldp.h`
`Ipcp`	Internet Protocol Control Protocol (over Ipv4)	`ipcp.h`
`Ipcp6`	Internet Protocol Control Protocol (over Ipv6)	`ipcp.h`
`Lcp`	Link Control Protocol	`lcp.h`
`Llc`	Logical Link Control Protocol	`llc.h`
`Llmnr`	Link-Local Multicast Name Resolution Protocol	`dns.h`
`Lqr`	Link Quality Report Protocol	`ppp.h`
`Ipv4`	Internet Protocol version 4	`ip.h`
`Ipv6`	Internet Protocol version 6	`ipv6.h`
`Msrpc`	Microsoft Remote Procedure Call Protocol	`msrpc.h`
`ModbusTCP`	Modbus over TCP Protocol	`modbus.h`
`Netbios`	Network Basic Input/Output System Protocol	`netbiosbase.h`

Protocol	Description	Protocol Definition File
NbtNs	NetBios Name Service	NbtNs.h
NbtNsOverTcp	NetBios Name Service protocol (over Tcp)	NbtNs.h
NbtSS	NetBios Session Service	NetBios.h
NbtDs	NetBios Datagram Service	NetBios.h
PPP	Point-to-Point Protocol	ppp.h
PPPoE	Point-to-Point Protocol over Ethernet	pppoe.h
SNAP	Standard Network Access Protocol	snap.h
Smb	Server Message Blocks	smb.h
Smb2	Server Message Blocks version 2	smb.h
SmbOverTcp	Server Message Blocks (over Tcp)	smb.h
Tcp	Transmission Control Protocol	tcp.h
Udp	User Datagram Protocol	udp.h

USB Monitoring Protocols

USB Monitor contains the following pre-defined protocols:

Video Class | USB Video Class | usb_video.h Audio Class | USB Audio Class | usb_audio.h Communication Class | USB Communication Class | usb_comm.h

In addition, it contains protocols for correctly decoding descriptor requests, including device, configuration and endpoint descriptors as well as full HID descriptor parsing.

Serial Monitoring Protocols

Serial Monitor contains the following pre-defined protocols:

MODBUS | Full definition of MODBUS RTU and MODBUS ASCII protocols modbus.h PPP | Point-to-Point Protocol | ppp.h

In addition, serial module uses protocols, defined in Network module through the PPP protocol.

Custom Protocols

In addition to built-in protocols, Device Monitoring Studio allows you to define custom protocols and use them for binding monitored data. After the custom protocol is defined and plugged into the protocol chain, it becomes available for all components, like Capture Filter, Display Filter, Structure View and others.

Device Monitoring Studio includes a built-in protocol editor, which simplifies developing and usage of custom protocols.

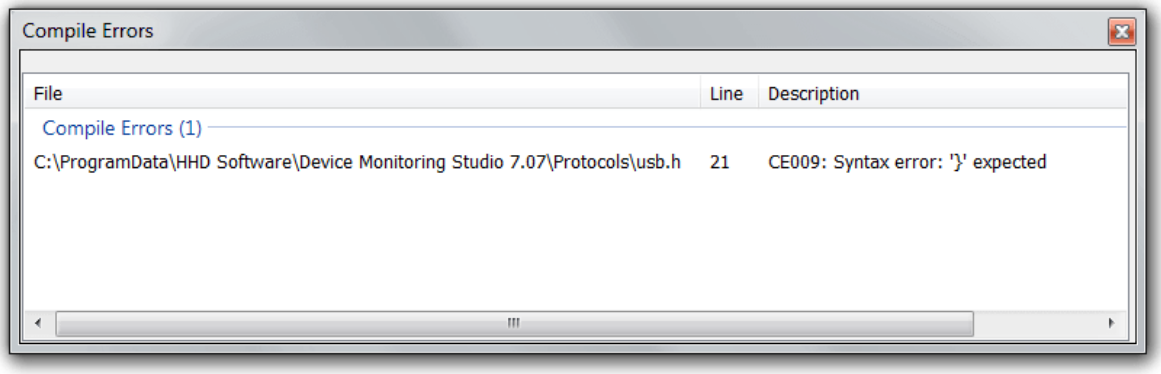
Basically, you follow this procedure to add a custom protocol(s) to Device Monitoring Studio:

1. Create a protocol definition file with one or more protocols defined. Use either built-in protocol editor or any external editor.
2. Edit the corresponding built-in protocol definition file to “plug” your new protocol(s) in the desired position in protocol chain.

See the Adding new Protocol Tutorial for more information.

Device Monitoring Studio will automatically recompile the whole protocol stack whenever you save your changes. This behavior may be switched off with a **Tools » Protocol Editor » Auto-recompile** command. You will be forced to stop any currently running monitoring sessions on every recompilation.

If there are any compilation errors, they will be displayed in **Compile Errors** tool window. Double-click on the error to open corresponding source file. Error position is automatically highlighted in a file.



Predefined Fields

Several predefined fields and functions are available for a custom protocol. This topic lists all those predefined components.

Common Predefined Identifiers

This section lists predefined fields available in all components: USB, Serial and Network:

`packet_ordinal`

A packet number (integer value). The very first packet has number 0. All subsequent packets have increasing numbers.

`entry_time`

Packet time stamp. This is the same format as in `FILETIME` structure (that is, it is a 64-bit unsigned integer representing the number of 100-nanosecond intervals since January 1, 1601). The time is UTC time.

`packet_size`

The size of the packet, including packet header.

`current_offset`

This pseudo field represents the offset from the beginning of the packet to the currently binding field. See current_offset for more information.

`device_source`

For multi-source session, this field contains the device ordinal (counting from zero). For other session types, this field is always zero.

Serial Monitor and Serial Bridge

Serial Monitor and Serial Bridge add the following predefined fields:

`packet_type`

A value from `PacketType` enumeration (std_serialdefs.h). May be one of the following values:

Constant	Description
`PACKET_CONDISCONNECT`	Port connection/disconnection packet.
`PACKET_GENERAL`	General packet. Contains either sent (written) or received (read) data.
`PACKET_CREATE`	Port open packet.
`PACKET_IO`	I/O Request packet.

`is_packet_up`

Boolean value which equals `true` if this packet is captured on its way *UP* or equals `false` if it is captured on its way *DOWN*.

`sending_device`

Used in Serial Bridge monitoring sessions. Equals to zero for first device and equals to one for the second device.

`status`

Holds the status of the I/O request. Zero means success, non-zero code is a `NTSTATUS` code, as defined in WDK.

The following global functions are available:

`get_communications_mode()`

Returns the current monitoring session's communication mode. See `ECommunicationType` enumeration defined in `std_serialdefs.h`. May be one of the following values:

Constant	Description
`GENERAL_COMMUNICATION`	No special communication mode is set.
`PPP_COMMUNICATION`	PPP communication mode is configured for a session.
`X0D_COMMUNICATION`	“One packet a line” communication mode is configured for a session.
`MODBUS_COMMUNICATION`	MODBUS communication mode is configured for a session.

`modbus_is_request_read()`

Returns `true` if “Parse requests on WRITE (responses on READ)” global setting is checked, `false` otherwise.

`modbus_get_use_rtu()`

Returns `true` if RTU mode is configured for MODBUS in settings, `false` otherwise.

USB Monitor

USB Monitor adds the following predefined fields and functions:

`event_type`

USB packet type. May be one of the following values:

Constant	Description
`EVENT_URB`	USB Request Block. Custom protocol should mainly focus on this packet type.
`EVENT_DEVICECONNECTED`	Device connection packet.
`EVENT_DEVICEDISCONNECTED`	Device disconnection packet.
`EVENT_DEVICESURPRISEREMOVAL`	Device surprise removal packet.
`EVENT_DEVICEQUERYID`	Device Query ID packet.
`EVENT_DEVICEQUERYTEXT`	Device Query Text packet.
`EVENT_PIPEINFO`	Internal packet used to report updated configuration descriptor to the USB Monitor. Should be ignored by custom protocol.
`EVENT_QUERYINTERFACE`	Device Query Interface packet.

`is_packet_up`

Boolean value which equals `true` if this packet is captured on its way *UP* or *false* if it is captured on its way *DOWN*.

`usb_get_vendor_name(vendorId)`

Resolves the USB vendor id to the vendor name. Returns a string.

`usb_get_model_name(vendorId, modelId)`

Resolves the USB model id to the device name. Returns a string.

Network Monitor

Network Monitor adds the following predefined fields:

`fragment_no`

Packet number for a fragment packet. If the current packet is not a fragment, this field equals to `LONG_MAX`.

`process_id`

ID of the process initiating the request.

`is_send_packet`

`true` for Send packets and `false` for Receive packets.

Protocol Reference

Serial Monitoring Sessions

Serial monitoring session uses the type `Serial`, declared in `serial.h` file to bind a serial packet. Please consult the file for detailed step-by-step description of binding process.

Serial Bridge Monitoring Sessions

Serial Bridge monitoring session uses the type `BridgePacket`, declared in `serial.h` file to bind a serial bridge packet. Please consult the file for detailed step-by-step description of binding process.

USB Monitoring Sessions

USB monitoring session uses the type `Usb`, declared in `usb.h` file to bind a USB packet. Please consult the file for detailed step-by-step description of binding process.

Network Monitoring Sessions

Network monitoring session uses the type `Ethernet`, declared in `network.h` file to bind a network packet. Please consult the file for detailed step-by-step description of binding process.

Protocol Editor

Built-in protocol editor allows you to view and edit predefined and custom protocol definition files. To open a file, use one of the following:

Protocols List Tool Window

Double-click on any file or protocol in a list to automatically open it in the editor.

Generic File » Open... command

Use this generic open command to open a protocol definition file.

Tools » Protocol Editor » Open Protocol File... command

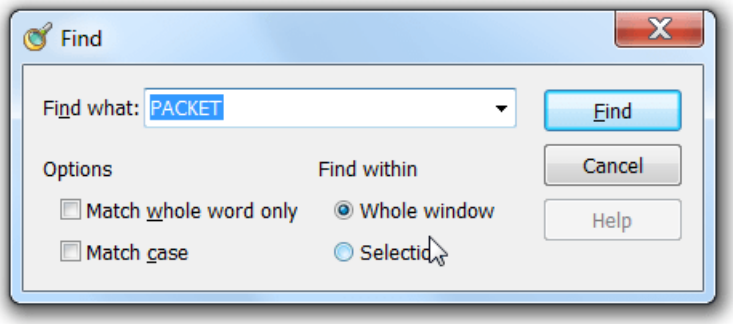
Use this command to open a protocol definition file.

Tools » Protocol Editor » New Protocol File...

Use this command to create a new protocol definition file and open it in the editor. You will be asked for a location and name.

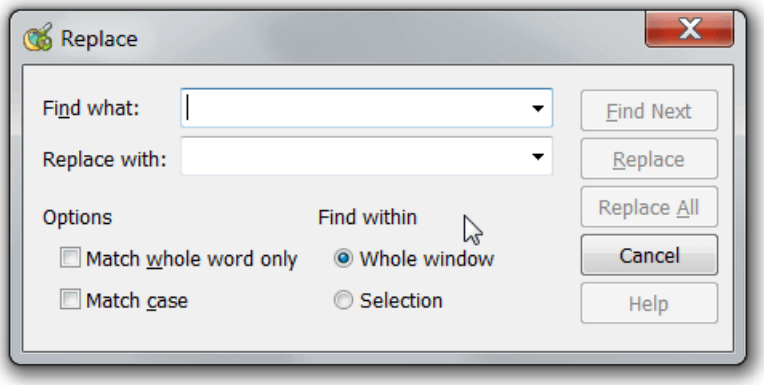
Editor uses syntax coloring (which you may customize as described in the Protocol Editor Coloring section) and provides advanced editing capabilities.

Find



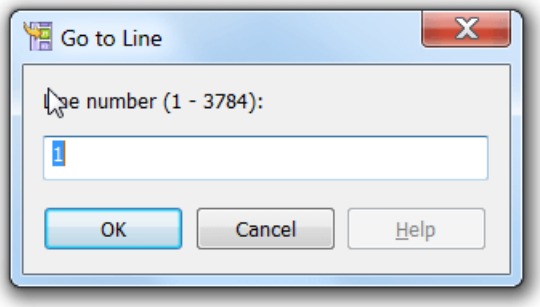
Use the **Edit » Find...** command to bring up the **Find** window. Enter the pattern to search, configure options and press the Find button. Then use the **Edit » Find Next** command to jump to the next pattern occurrence.

Replace



Use the **Edit » Replace...** command to bring up the **Replace** window. Enter the source and replacement patterns, configure options and then press the **Replace** or **Replace All** button.

Go to Line



Use the **Edit » Go to Line** command to bring up the **Go to Line** window. Enter the line number and press **OK** button.

Protocols List Tool Window

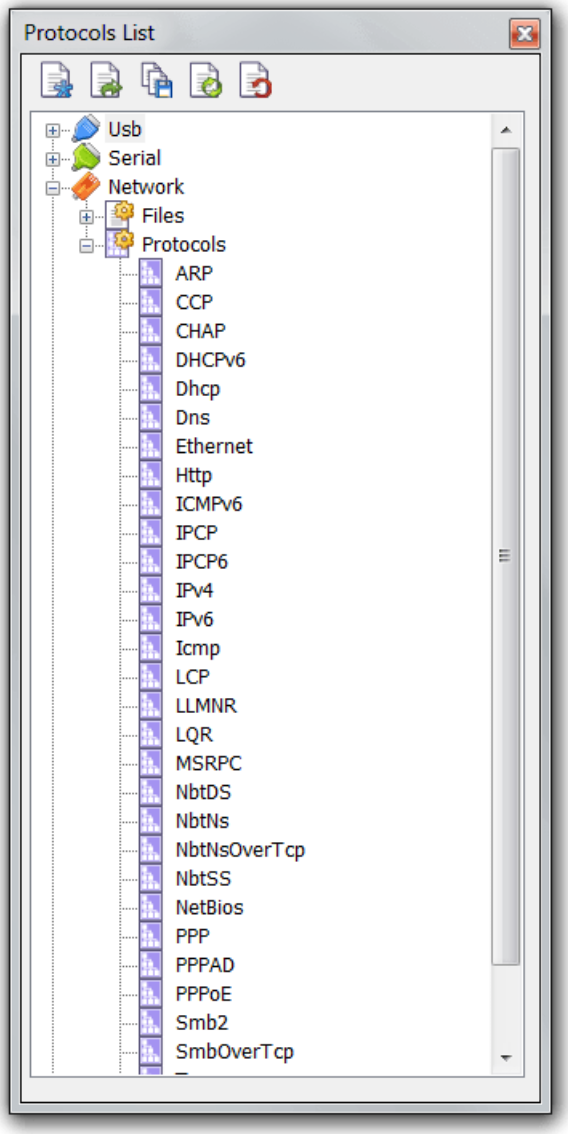
Protocols List tool window displays all loaded protocol files as well as list of protocols defined in them. At the root level, all installed modules are displayed, such as Serial, USB and Network. For each, files and protocols are displayed in corresponding groups. Double-click on the file to open this file in the editor. Double-click on the protocol to open its corresponding file and automatically navigate to protocol definition.

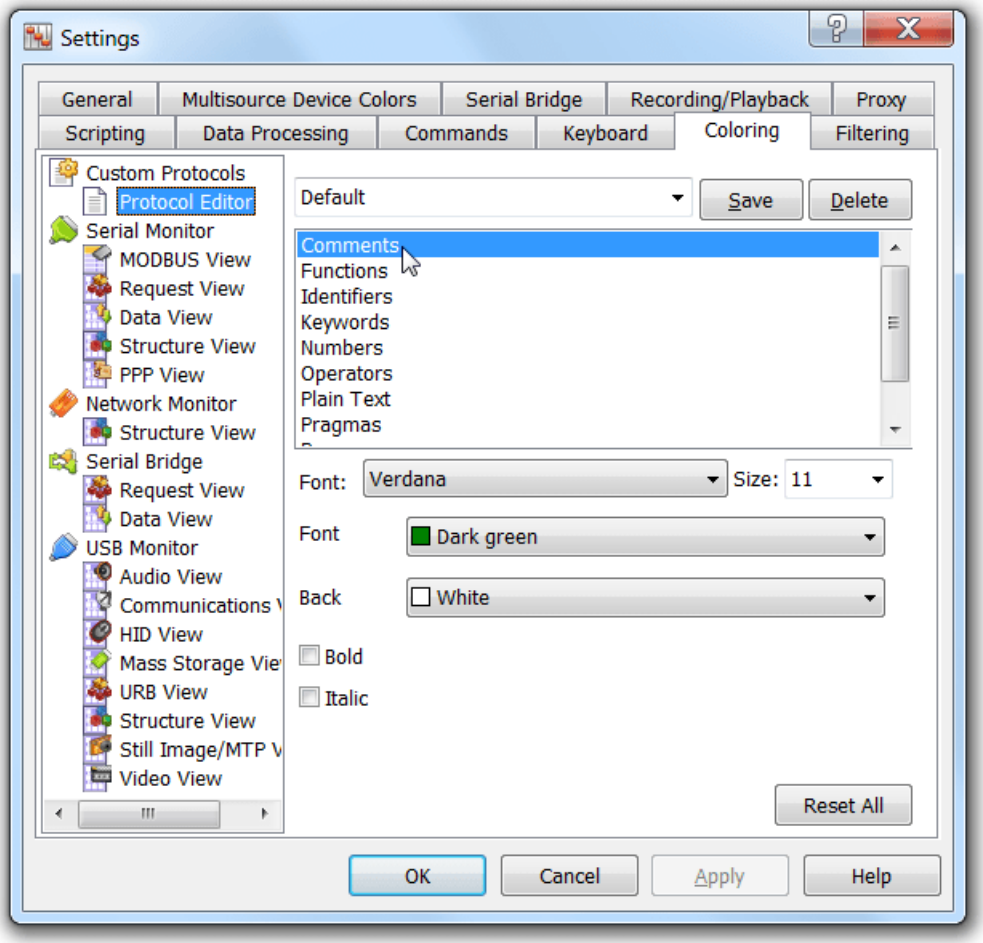
Unknown Files

Device Monitoring Studio automatically scans the protocol folder (`%PROGRAMDATA%\HHD Software\Device Monitoring Studio x.xx`) for protocol definition files. If it finds files not referenced by any protocol chain, it will include them into **Unknown** group. As soon as you plug your protocol definition to some of protocol chains, it will be moved into corresponding group.

You may delete any file in the Unknown group by using the **Tools » Protocol Editor » Delete File** command.

Coloring


Protocols List
Usb
Serial
Network
Files
Protocols
ARP
CCP
CHAP
DHCPv6
Dhcp
Dns
Ethernet
Http
ICMPv6
IPCP
IPCP6
IPv4
IPv6
Icmp
LCP
LLMNR
LQR
MSRPC
NbtDS
NbtNs
NbtNsOverTcp
NbtSS
NetBios
PPP
PPPAD
PPPoE
Smb2
SmbOverTcp



This window allows you to configure fonts and colors used by protocol editor. You can change the appearence of the following categories:

Comments

Comments are standard C/C++ comments sections, enclosed between /* and */ or one-line commentaries started with //

Functions

Functions declared in protocol definition files

Identifiers

Identifiers, such as field names, variable names, protocol and structure names.

Keywords

Protocol definition language keywords, such as statements and types.

Numbers

All numeric values.

Operators

All supported operators.

Plain text

Other un-categorized text.

Pragmas

Pragma declarations.

Preprocessor

Preprocessor directives.

Strings

String literals.

Licenses

This product includes the open source project Scintilla. Scintilla project license is provided below and is also available at http://www.scintilla.org/License.txt

License for Scintilla and SciTE

Tutorials

Adding New Protocol

This section will guide you through creation of your own protocol and adding it to the network protocol chain. You may use the same steps to create and add protocols to other protocol chains.

Creating New Protocol Definition File

First, we will create an empty protocol definition file. Execute the **Tools » Protocol Editor » New Protocol File** command. You will be presented with a standard Windows Save File dialog. Do not change the location and enter the following name: `mygame.h`. An empty editor file will be opened.

Copy and paste the following text into the protocol definition file:

```C++
//Custom protocol
//Network Game Sample

// Include standard headers
#include "std_netdefs.h"

#pragma pack(1)

enum MessageType : DWORD
{
  MSGID_CONNECT_PLAYER = 1,
  MSGID_DISCONNECT_PLAYER,
  MSGID_CREATE_PLAYER,
  MSGID_SEND_PLAYER,
  MSGID_SEND_PLAYER_POS,
  MSGID_SEND_PLAYER_CHARS,
  MSGID_LAST
```

```
};

struct pstring_a
{
  BYTE length;
  char text[length];
};

struct MyCreatePlayer
{
  pstring_a PlayerName;
};

struct MySendPlayerAdded
{
  DWORD PlayerID;
};

struct MyConnect
{
  DWORD PlayerID;
};

struct MyDisConnect
{
  DWORD PlayerID;
};

struct MySendPlayerPos
{
  float x;
  float y;
  float z;
};

struct MySendPlayerCharacteristics
{
  WORD Strength;
  WORD Dexterity;
  WORD Health;
  WORD Mana;
};

[category(protocol)]
public struct MyGameProtocol
{
  MessageType MsgID;
  WORD Version;
  switch(MsgID)
  {
    case MSGID_CONNECT_PLAYER:
      MyConnect connect;
      break;
    case MSGID_DISCONNECT_PLAYER:
      MyDisConnect disconnect;
      break;
    case MSGID_CREATE_PLAYER:
      MyCreatePlayer create;
      break;
    case MSGID_SEND_PLAYER:
      MySendPlayerAdded send_added;
      break;

    default:
      if(MsgID > MSGID_SEND_PLAYER && MsgID < MSGID_LAST)
      {
        struct GameData
        {
          DWORD PlayerID;
          switch(MsgID)
          {
            case MSGID_SEND_PLAYER_POS:
              MySendPlayerPos player_pos;
              break;
            case MSGID_SEND_PLAYER_CHARS:
              MySendPlayerCharacteristics player_chars;
```

```
                MySendPlayerCharacteristics player_chars;
                break;
            }
        } game_data;
    }
    else
    {
      BYTE Unknown[packet_size - current_offset];
    }
  }
};
```

Save the file using the **File » Save** command.

Plugging New Protocol to Protocol Chain

Next step is to plug our new protocol into the protocol chain. We need to tell Device Monitoring Studio where and under what conditions this new protocol should be bound to monitored data.

Our game uses UDP for transport and has a specific UDP port. We will use this information to plug our game protocol. First, use the Protocols List tool window to open the UDP protocol. Expand the "Network" category, then "Protocols" category and double-click on the "Udp" item.

As you see, part of the `Udp` protocol definition includes a `switch` that selects next enclosed protocol. Modify the switch by adding the following text just before the `default:` line:

```
C++
case MyPort:
  MyGameProtocol my_proto;
  break;
```

Then, scroll to the beginning of the `udp.h` file and add the following text (after all `#include` directives):

```
C++
// Include the contents of the protocol definition file
#include "mygame.h"

// Define the port our game is using
const MyPort = 32323;
```

As soon as you save the `udp.h` file (using the **File » Save** command), Device Monitoring Studio will automatically compile the changes. If there are any compilation errors, they will be displayed in the Compile Errors tool window. Otherwise, next network monitoring session you start will automatically use your new protocol whenever UDP packet to port 32323 is captured.

Language Reference

Device Monitoring Studio supports advanced structure definition syntax. It is based on the Standard C type definition syntax and extends it in a number of ways.

While Standard C only allows defining *static* types, that is, data structures which size and memory allocation is strictly defined at compile time, Device Monitoring Studio extends the syntax to allow definition of *dynamic* types. The following code illustrates this:

```C++
struct StaticType
{
    int Array[128];
};

struct DynamicType
{
    int ArraySize;

    // invalid in Standard C (non const expression), valid in Device Monitoring Studio
    int Array[ArraySize / sizeof(int)];
};
```

The `DynamicType` structure definition is valid in Device Monitoring Studio. The size of the structure is determined at the time it is *bound* to the data and the number of elements in `Array` array varies.

Workflow

Device Monitoring Studio compiles all protocol definition files in its library on each application start. A preprocessor is run on each source file. It is responsible for the following tasks:

- Elimination of all comments from the source file.
- Macro evaluation.
- Processing of conditional compilation blocks.
- Processing of all `#include` directives and building the dependency graph.

Multiple files are pre-processed and compiled in parallel if Device Monitoring Studio is running on multi-processor and/or multi-core computer.

Tokenization

Source files are tokenized according to standard C language rules: there must be one or more space characters between tokens if they cannot be distinguished without spaces and there may optionally be one or more spaces between tokens if they are distinguishable without spaces.

Space is a space character (`' '`), tabulation (`'\t'`), newline (`'\n'`) or comment.

```C++
int a;              // valid, space is used to separate tokens "int" and "a"
int/* comment */a;  // valid, comment  is used to separate tokens "int" and "a"
inta;               // invalid, compiler cannot distinguish between "int" and "a" and parses it
as "inta"
int                 // continued on the next line…
     a              // still continued…
          ;         // valid, newline is valid space character

a & b               // valid, space is used to separate "a" and "&" and "&" and "b"
a&b                 // still valid, space is not required - compiler is able to
                    // distinguish between "a" (identifier) and "&" (operator)
```

Space must not be used inside a keyword, such as built-in type `int` or operator `&&`:

```C++
in t a;       // invalid, "int" must not contain space
a & & b       // valid, but will be parsed as a & (&b)
a | | b       // invalid, space inside keyword (operator "||")
```

Comments

Two standard C-style comments are supported: single-line and multi-line comments. Single-line comment starts with `//` character sequence and continues to the end of the current line. Multi-line

comments must be started with `/*` sequence and terminated with `*/` sequence.

```
C++
// Single-line comment
/* multi-line
  (continued here)
  comment */
```

Multi-line comments may be used “in-place”:

```
C++
struct /* comment */ A
{
    int /* another comment */ a;
};
```

Comments are completely ignored and removed from the document prior to compiling it.

Preprocessor

Preprocessor is a special compiler that is run each time the application compiles the protocol definition file. It executes before compilation of the source file and prepares a source file for compilation.

Device Monitoring Studio provides a fully C99-compliant preprocessor which supports the following directives:

`#include`

Performs a physical inclusion of the contents of another source file into the current file.

`#pragma once`

Prevents a file from being included multiple times.

`#define`, `#undef`

Allows defining preprocessing constants and macros.

`#if`, `#ifdef`, `#ifndef`, `#else`, `#elif`, `#endif`, `defined() operator`.

Provides support for conditional compilation.

`#error`

Unconditionally stops source file compilation.

#include directive

`#include` directive physically includes the contents of a specified file into the current file.

Syntax:

```
C++
#include "filename"
```

or

```
C++
#include <filename>
```

When used in its first form, the referenced file is searched in the same folder as the current file.

For example, let us have following two files:

`file1.h`

```C++
struct A
{
    // …
};
```

And `file2.h`:

```C++
#include "file1.h"

struct B
{
    A a;
};
```

If there was no `#include` directive in `file2.h`, you would not be able to add this file to a Structure Library, as it uses an undefined type `A`. But preprocessor, which runs on the file before it is compiled, transforms the file into the following:

```C++
struct A
{
    // …
};

struct B
{
    A a;
};
```

That is, it physically inserts the contents of `file1.h` into `file2.h`, thus, making `file2.h` compilable. See also the `#pragma once` directive.

Using Absolute and Relative Paths

Both syntax forms, form 1 and form 2 allow you to specify absolute or relative paths, for example:

```C++
// will use an absolute path
#include "c:\Projects\definitions.h"

// includes "definitions.h" file, located in "inc" sibling
#include "..\inc\definitions.h"

// includes "definitions.h" file, located in "lib" subdirectory
// of one of standard include paths.
#include <lib\definitions.h>
```

#pragma once Directive

`#pragma once` directive prevents the file from being included multiple times.

Syntax:

```C++
#pragma once
```

Let `fileA.h` be `#include`-ed into `fileB.h` and into `fileC.h`. If you now write a `fileD.h` with the following two lines:

```C++
#include "FileB.h"
#include "FileC.h"
```

“fileA.h” will get included twice. In order to prevent this, put a following line into the “fileA.h” file:

```C++
#pragma once
```

#define Directive

`#define` directive lets you define preprocessor-time constants and macros.

Syntax:

```C++
#define id [token-string]
```

or

```C++
#define id([id, [id, […]]) [token-string]
```

A macro declaration must end on the same line. If you need to continue on the next line, finish a line with a backslash character:

```C++
#define MY_STRING "This is a start of the long "   \
          "long long " \
          "string"    // the end of macro declaration
```

Defining Constants

In its first form, #define creates a preprocessor-time constants. For example:

```C++
#define MAX_LENGTH 5
```

This declaration will instruct the preprocessor to scan the source file and replace all occurrences of `MAX_LENGTH` with `5`. The preprocessor is smart enough to only perform a replace when `MAX_LENGTH` is used as an identifier, that is, it will transform the following code fragment:

```C++
// This example uses MAX_LENGTH:
struct A
{
    int array[MAX_LENGTH];
    const int MaxLength = MAX_LENGTH;
    $assert(MaxLength == MAX_LENGTH, "Error with MAX_LENGTH");
};
```

into the following:

```C++
// This example uses MAX_LENGTH:
struct A
{
    int array[5];
    const int MaxLength = 5;
    $assert(MaxLength == 5, "Error with MAX_LENGTH");
};
```

As you see, substitution had not occurred in comment and in string. All other occurrences have been replaced.

A constant may be more complex, for example:

```C++
#define MAX_LENGTH (2 + 5)
```

Note the use of parenthesis in order to prevent operator precedence errors.

Defined constants may refer to constants defined before:

```C++
#define SECOND 10000000
#define MINUTE (60 * SECOND)
```

The following form:

```C++
#define SOME
```

will only define a constant, without assigning it a particular value. If occurred in a text, it gets removed from it. However, you may successfully test such constant within the `#ifdef` directive or using a `defined()` operator:

```C++
#define INCLUDE_STRUCT_A
// …
#ifdef INCLUDE_STRUCT_A
struct A
{
    // …
};
#endif
```

By convention, macros and preprocessor constants are named with uppercase letters.

Defining Macros

The second form of the directive is used to define macros:

```C++
#define ADD(x,y) ((x) + (y))
```

The preprocessor will not only perform a replace, but also will perform a parameter substitution:

```C++
ADD(7,8)        // will be replaced with ((7) + (8))
ADD(n,-1)       // will be replaced with ((n) + (-1))
```

Macros may contain any number of parameters:

```C++
#define MAX(a,b) ((a) > (b) ? (a) : (b))
```

Variadic Macros

Device Monitoring Studio supports variadic macros. To use variadic macros, the ellipsis may be specified as the final formal argument in a macro definition, and the replacement identifier `__VA_ARGS__` may be used in the definition to insert the extra arguments. `__VA_ARGS__` is replaced by all of the arguments that match the ellipsis, including commas between them.

#undef Directive

An `#undef` directive is used to remove the macro definition.

Syntax:

```C++
#undef id
```

Example:

```C++
#define MAX_LENGTH 5
struct A
{
    int array[MAX_LENGTH];
};

#undef MAX_LENGTH
struct B
{
    // compile-time error, "MAX_LENGTH" identifier not found
    // (preprocessor did not replace it with "5")
    int array[MAX_LENGTH];
};
```

#error Directive

`#error` directive stops compilation of the current source file.

Syntax:

```C++
#error error-message-string
```

`error-message-string` is displayed to the user.

Preprocessor Operators

Stringizing Operator

The number-sign or "stringizing" operator `#` converts macro parameters to string literals without expanding the parameter definition. It is used only with macros that take arguments. If it precedes a formal parameter in the macro definition, the actual argument passed by the macro invocation is enclosed in quotation marks and treated as a string literal. The string literal then replaces each occurrence of a combination of the stringizing operator and formal parameter within the macro definition.

White space preceding the first token of the actual argument and following the last token of the actual argument is ignored. Any white space between the tokens in the actual argument is reduced to a single white space in the resulting string literal. Thus, if a comment occurs between two tokens in the actual argument, it is reduced to a single white space. The resulting string literal is automatically concatenated with any adjacent string literals from which it is separated only by white space.

Further, if a character contained in the argument usually requires an escape sequence when used in a string literal (for example, the quotation mark (") or backslash () character), the necessary escape backslash is automatically inserted before the character.

```
C++
#define ASSERT(condition, message) $assert(condition, message ": " #condition)

// …
struct A
{
    int a;
    ASSERT(a > 5, "a must be greater than 5");
// The previous line will be replaced with $assert(a > 5, "a must be greater than 5" ": " "a >
5");
};
```

Token-Pasting Operator

The double-number-sign or “token-pasting” operator `##`, which is sometimes called the “merging” operator, is used in both object-like and function-like macros. It permits separate tokens to be joined into a single token and therefore cannot be the first or last token in the macro definition.

If a formal parameter in a macro definition is preceded or followed by the token-pasting operator, the formal parameter is immediately replaced by the un-expanded actual argument. Macro expansion is not performed on the argument prior to replacement.

Then, each occurrence of the token-pasting operator in token-string is removed, and the tokens preceding and following it are concatenated. The resulting token must be a valid token. If it is, the token is scanned for possible replacement if it represents a macro name. The identifier represents the name by which the concatenated tokens will be known in the program before replacement. Each token represents a token defined elsewhere, either within the program or on the compiler command line. White space preceding or following the operator is optional.

This example illustrates use of token-pasting operator:

```
C++
#define BIT_FIELD(type, n)  type field##n:n
// …
struct A
{
    BIT_FIELD(unsigned,5); // will be replaced with: unsigned field5:5;
    BIT_FIELD(unsigned,7); // will be replaced with: unsigned field7:7;
};
```

Conditional Compilation Directives

`#if`, `#elif`, `#else`, `#endif`, `defined() operator`

The `#if` directive, with the `#elif`, `#else`, and `#endif` directives, controls compilation of portions of a source file. If the expression you write (after the `#if`) has a nonzero value, the line group immediately following the `#if` directive is retained in the translation unit.

```
C++
#if expression
text
[
#elif expression
text
[
#elif expression
text
]]
[
#else
text
]
#endif
```

Each `#if` directive in a source file must be matched by a closing #endif directive. Any number of `#elif` directives can appear between the #if and #endif directives, but at most one #else directive is allowed. The #else directive, if present, must be the last directive before `#endif`.

The `#if`, `#elif`, `#else`, and `#endif` directives can nest in the text portions of other #if directives. Each nested `#else`, `#elif`, or `#endif` directive belongs to the closest preceding `#if` directive.

All conditional-compilation directives, such as `#if` and `#ifdef`, must be matched with closing `#endif` directives prior to the end of file; otherwise, an error message is generated. When conditional-compilation directives are contained in include files, they must satisfy the same conditions: There must be no unmatched conditional-compilation directives at the end of the include file.

Macro replacement is performed within the part of the command line that follows an `#elif` command, so a macro call can be used in the expression.

The preprocessor selects one of the given occurrences of text for further processing. A block specified in text can be any sequence of text. It can occupy more than one line. Usually text is program text that has meaning to the compiler or the preprocessor.

The preprocessor processes the selected text and passes it to the compiler. If text contains preprocessor directives, the preprocessor carries out those directives. Only text blocks selected by the preprocessor are compiled.

The preprocessor selects a single text item by evaluating the constant expression following each `#if` or `#elif` directive until it finds a true (nonzero) constant expression. It selects all text (including other preprocessor directives beginning with #) up to its associated `#elif`, `#else`, or `#endif`.

If all occurrences of constant-expression are false, or if no `#elif` directives appear, the preprocessor selects the text block after the `#else` clause. If the `#else` clause is omitted and all instances of constant-expression in the `#if` block are false, no text block is selected.

The constant-expression is an integer constant expression with these additional restrictions:

- Expressions must have integral type and can include only integer constants, character constants, and the `defined()` operator.
- The expression cannot use sizeof() operator.

The preprocessor operator `defined` can be used in special constant expressions, as shown by the following syntax:

```
C++
defined(identifier)
defined identifier
```

This constant expression is considered true (nonzero) if the identifier is currently defined; otherwise, the condition is false (0). An identifier defined as empty text is considered defined. The `defined` directive can

be used in an `#if` and an `#elif` directive, but nowhere else.

```C++
// Define the structure definition variant:
#define VARIANT 2

// …

struct A
{
#if VARIANT == 1
    char a;
#elif VARIANT == 2
    short a;
#elif VARIANT == 3
    int a;
#else
    long a;
#endif
};
```

`#ifdef`, `#ifndef`

The `#ifdef` and `#ifndef` directives perform the same task as the `#if` directive when it is used with `defined(identifier)`.

```C++
#ifdef identifier
#ifndef identifier

// equivalent to
#if defined(identifier)
#if !defined(identifier)
```

You can use the `#ifdef` and `#ifndef` directives anywhere `#if` can be used. The `#ifdef` identifier statement is equivalent to `#if 1` when identifier has been defined, and it is equivalent to `#if 0` when identifier has not been defined or has been undefined with the `#undef` directive. These directives check only for the presence or absence of identifiers defined with `#define`.

Predefined Macros

Device Monitoring Studio defines several macros, called predefined macros. These macros are available for each structure definition file:

Macro	Description
`_SVC_VER`	An integer that specifies the current compiler's version. Low 8 bits represent the minor version, high 8 bits represent major version. For example, version 3.05 is represented as 0x305.
`_SVC_X86`	Defined for 32-bit version of Device Monitoring Studio. Undefined for 64-bit version.
`_SVC_X64`	Defined for 64-bit version of Device Monitoring Studio. Undefined for 32-bit version.
`_SVC_POINTER_SIZE`	An integer that specifies the size of pointer in bits on current machine. Equals to 32 for 32-bit version of Device Monitoring Studio and 64 for 64-bit version.
`_SVC_POINTER_SIZE_REAL`	An integer that specifies the size of pointer in bits on current machine. Always returns the real value, even if 32-bit version of Device Monitoring Studio is running on 64-bit OS.

Built-in Types

Integer Types

Device Monitoring Studio natively supports the following integer types:

Type Name	Other Valid Names	Typedefs in stddefs.h	Description	Range
`bool`	N/A	N/A	One-byte boolean value. Displayed as "true" and "false".	N/A
`char`	`signed char`	CHAR, int8, __int8	8-bit signed character	-2^7 to $2^7 - 1$
`wchar_t`	N/A	`WCHAR`	16-bit UNICODE character	0 to $2^{16} - 1$
`unsigned char`	N/A	`UCHAR`, `BYTE`, `uint8`	8-bit unsigned character	0 to $2^8 - 1$
`short`	`signed short`, `short int`, `signed short int`	`SHORT`, `int16`, `__int16`	16-bit signed integer	-2^{15} to $2^{15} - 1$
`unsigned short`	`unsigned short int`	`USHORT`, `WORD`, `uint16`	16-bit unsigned integer	0 to $2^{16} - 1$
`int`	`signed int`, `signed`	`INT`, `int32`, `__int32`	32-bit signed integer	-2^{31} to $2^{31} - 1$
`unsigned int`	`unsigned`	`UINT`, `uint32`	32-bit unsigned integer	0 to $2^{32} - 1$
`long`	`signed long`, `long int`, `signed long int`	`LONG`	32-bit signed integer	-2^{31} to $2^{31} - 1$
`unsigned long`	`unsigned long int`	`ULONG`, `DWORD`	32-bit unsigned integer	0 to $2^{32} - 1$
`__int64`	N/A	`LONGLONG`, `int64`	64-bit signed integer	-2^{63} to $2^{63} - 1$
`unsigned __int64`	N/A	`ULONGLONG`, `FILETIME`, `uint64`	64-bit unsigned integer	0 to $2^{64} - 1$

Type Modifiers

Two modifiers, `little_endian` and `big_endian` may be used to change the default byte ordering of a type. They may be used directly in declaring fields:

```
C++
struct A
{
    big_endian short value;
};
```

or in typedef declaration to create a new type:

```
C++
typedef big_endian short b_short;
```

Note that #pragma byte_order directive may still be used to change default byte ordering for the rest of a scope. If `little_endian` or `big_endian` modifier is present, it always overwrites the current default.

Floating-Point Types

Device Monitoring Studio natively supports the following floating-point types:

Type Name	Description	Size, in Bytes	Range
`float`	Single-precision floating-point type, according to IEEE 754-1985	4	$3.4 \cdot 10 \pm^{38}$ (7 digits)
`double`	Double-precision floating-point type, according to IEEE 754-1985	8	$1.7 \cdot 10 \pm^{308}$ (15 digits)

String Types

Device Monitoring Studio natively supports the following string types:

Type Name	Description
`string`	Null-terminated ANSI string (each character occupies single byte).
`wstring`	Null-terminated UNICODE string (each character occupies two bytes).

Expressions

Expressions are allowed in different places in the structure definition. For example, the size of the bit field, the number of items in array and pointer offset are all specified using expressions. Expressions are also used in calculating constant values and enumeration values.

Expression is a combination of immediate values, constants, enumeration values, function calls and field references. All these elements are connected with one or more operators.

An expression may be as simple as:

```
C++
5  // evaluates to integer "5"
```

or as complex as:

```
C++
5 + 7*(10 - 2)    // calculate an expression
info.bmiHeader.sel.header.biSizeImage  // take a value of a field several scopes deep
bfTypeAndSignature.bfType == 'BM' && bfReserved1 == 0 && bfReserved2 == 0   // verify a
condition
RvaToVa(OptionalHeader.DataDirectory[i].VirtualAddress)  // access a field in a nested structure
and array and call an external function
```

Operators

Below is a table of supported operators. Operators are sorted by their precedence, from highest to lowest. Operators in the same row have the same precedence value and are evaluated from left to right.

Operator(s)	Name or Meaning
`.` `[]` `()`	Field access, array indexing, expression grouping
`()`	Function call
`-` `~` `!` `sizeof()` `&`	Unary minus, bitwise NOT, logical NOT, sizeof, address-of
`*` `/` `%`	Multiplication, division, modulo division
`+` `-`	Addition, subtraction
`<<` `>>` `>>>`	Left shift, right shift, right unsigned shift
`<` `<=` `>` `>=`	Less than, less than or equal, greater than, greater than or equal
`==` `!=`	Equality, inequality
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`&&`	Logical AND
`||`	Logical OR
`?:`	Conditional operator

Optimization

All expressions are evaluated at the time a structure file is compiled. If expression is successfully evaluated to constant value (that is, it does not contain any field references), calculated value is used instead of the expression. It is used each time a type is bound to the data, thus greatly minimizing bind time. You can get advantage of this optimization taken by the Device Monitoring Studio: instead of

```
C++
const SecondsInHour = 3600;    // 60 * 60
const SecondsInDay = 86400;    // 60 * 60 * 24
```

use

```
C++
const SecondsInHour = 60*60;
const SecondsInDay = SecondsInHour * 24;
```

As the result will be the same after the source file is compiled.

Device Monitoring Studio is also capable of optimizing sub-expressions:

```
C++
struct A
{
    int size;
    byte data[size * (sizeof(int) - 1)];
};
```

Important note here is that constant sub-expression must be enclosed in parenthesis in order to be optimized. Otherwise, Device Monitoring Studio will not optimize it because it considers a variable as having arbitrary type:

```C++
var i = ...
var j = i + (5 + 3);    // will be optimized to i + 8
var j = i + 5 + 3;    // will not be optimized (consider the case when i is a string, for example, which results in i + "53").
```

Immediates

An expression may contain an *immediate value*. There are several ways to specify an immediate:

Integer Number

A signed or unsigned integer number may be specified in either base 10 or base 16:

```C++
9    // integer number 9
-7   // integer number -7
0x5a // integer number 90
```

Floating-Point Number

A floating-point number is specified in either standard or scientific notation:

```C++
2.3    // floating-point number 2.3
-4e-9  // floating-point number -0.000000004
```

Character Constants

A character constant is one, two or four symbols enclosed in single quotation marks:

```C++
'c'      // integer number 99
'BM'     // integer number 19778
'NTFS'   // integer number 1397118030
```

A standard C escape character may be used inside the quotation marks:

```C++
'\n'   // integer number 10
'\t'   // integer number 9
```

String Constants

String constants are only allowed in $assert directive and in $print directive. A sequence of characters (including escape characters) enclosed in double quotation marks. Several subsequent strings are automatically concatenated.

```C++
"Hello, World!"              // "Hello, World!" string
"Hello, " "World!"           // the same "Hello, World!" string (automatically concatenated)
"Col1\tCol2\tCol3\n1\t2\t3"   // simple table with two rows and three columns
```

References

Device Monitoring Studio allows getting a reference to a field by using the built-in function ref. The following operators are allowed for two references:

- == operator
- != operator
- < operator
- <= operator
- > operator
- >= operator

If one argument of a binary operator is a reference and another one is not, a reference is automatically dereferenced and then the normal operator rules apply.

In addition, a reference may be dereferenced using one of the built-in conversion functions.

Limitations

The result of ref function application may not be used with [] operator and . operator. Assign the result of the function to the local variable and use the variable instead:

```
C++
struct A
{
    struct { int a,b; } field;

    var t1 = ref(field).a;    // syntax error
    var r_field = ref(field);
    var t2 = r_field.a;       // OK

    int array[100];
    var t3 = ref(array)[50];  // syntax error
    var r_array = ref(array);
    var t4 = r_array[50];     // OK
};
```

Byte Arrays

Byte arrays are introduced in version 5.15. This is a new value type in addition to previously supported `boolean`, `integer`, `floating-point`, `string` and reference value types.

Byte arrays represent contiguous ranges in memory and usually contain unstructured data. The following operations are supported on byte arrays:

- References are implicitly converted to byte arrays if they are used in the expression with other byte arrays. See example below.
- Byte arrays may be specified as immediate values using the following syntax:

```
C++
struct Header
{
    unsigned char Signature[3];

    $assert(Signature == { 0x20, 0xff, 0xfe }); // Immediate byte array
    // Note the implicit conversion between the array field and byte array.
    // Array field is first automatically converted to reference and then implicitly
converted to
    // byte array
};
```

- Other value types may be explicitly converted to byte array using the array() function.

- Arrays may be compared for equality and inequality. Arrays also support the following operators: `<`, `>`, `<=` and `>=` for which lexicographical comparison is used.
- An individual element of byte array may be taken using the element() function.
- A portion of an array may be extracted using the subarray() function.
- A length() function returns the number of bytes in an array. This function also works with strings, returning the length of the string.
- A find() function allows you to search for an occurrence of one byte array within another one. This function also works with strings.

Field Access

An expression may reference any field within visible scope. A field is referenced by its name, which is case sensitive. Name lookup continues from child scope to parent scope until the match is found. If the field is not found, constants and enumeration values are searched. If no match is found in constants and enumeration values, an error is generated.

The following example illustrates this:

```
C++
enum
{
    EnumerationValue      =0x20,
};

const ConstantValue = 0x30;

struct A
{
    int a;
    int Array1[a];                 // references a in this structure
    int Array2[ConstantValue];     // references ConstantValue in global scope
    int Array3[EnumerationValue];  // references EnumerationValue in global scope
    struct
    {
        int Array4[a];             // references a in parent scope
    } contained;
};
```

Pointed to types are allowed to reference fields in pointer field's scope. It is their direct enclosing scope. For example:

```
C++
struct PointedType;     // forward declaration

struct PointerType
{
    int a;
    int b as PointedType *;    // b is converted to a pointer to PointedType structure
    int c;
};

struct PointedType
{
    int Array[a + c];        // references a and c in PointerType (note c is declared AFTER b,
the reference is still valid)
};
```

`this` Pseudo-Field

The special field this is defined for each bound user-defined type and evaluates to the absolute offset of

this bound structure.

```
C++
struct B
{
    // …
};

struct A
{
    int OffsetToB as B *(this);    // B will automatically be bound to &A + OffsetToB
};
```

When passed to the built-in `ref` function, the result is a `reference` to the current object.

`array_index` Built-In Variable

This built-in variable evaluates to the current array element index, if the structure is being used inside an array, otherwise it is evaluated to -1.

```
C++
struct B;

struct A
{
    B b[10];
};

struct B
{
    int a;
    if (array_index == 3)    // array_index evaluates to a current array index
        int b;
};
```

`current_offset` Built-In Variable

This built-in variable evaluates to the address the next field will be bound to.

```
C++
#pragma script("get_doc_size.js")    // we use a GetDocumentSize() function

struct A
{
    int b[10];
// pad this structure to the end of the file
    char padding[GetDocumentSize() - current_offset];
};
```

. Field Access Operator

Field access operator is used to access fields in contained scopes.

Syntax:

```
C++
id.id[.id[.id …]]
```

The following code fragment illustrates the use of the field access operator:

```C++
struct A
{
    struct
    {
        int c;
    } b;
    int Array[b.c];     // access the c field inside the b field
};
```

struct, union, case union or pointer field may be used on the left of the field access operator.

[] Array Indexing Operator

Array indexing operator allows you to reference individual array items.

Syntax:

```C++
id[exp]
```

The following code fragment illustrates the use of the array indexing operator:

```C++
struct A
{
    int Array1[10];         // static array of 10 integers
    int Array2[Array1[2]];  // dynamic array, which size is taken from the third value of Array1
array
};
```

Array elements are numbered from 0 to `Count` - 1, where `Count` is (static or dynamic) array size.

() Expression Grouping Operator

Expression grouping operator allows you to change the precedence rules of operators in an expression.

Syntax:

```C++
(exp)
```

The following code fragment illustrates the use of the expression grouping operator:

```C++
2 + 3 * 4 - 7       // evaluates to 7
2 + 3 * (4 - 7)      // evaluates to -7
```

You can use parenthesis whenever you need to change the precedence of operators or increase the readability of the expression.

() Function Call Operator

Function call operator allows you to call an internal or external function.

Syntax:

```C++
id( [param-list] )
param-list: expr [,expr ,[…] ]
```

The following code fragment illustrates the use of the function call operator:

```C++
struct A
{
    int Array1[int(10.2)];         // Call an internal cast function
    int Array2[ExternalFunction()];  // Call an external function with no parameters
};
```

External function calls are never optimized at compile-time. All expressions containing external function calls will be computed at run-time.

- Unary Minus Operator

Unary minus operator change the sign of the expression.

Syntax:

```C++
-exp
```

The following code fragment illustrates the use of the unary minus operator:

```C++
-7       // evaluates to integer number -7
```

~ Bitwise NOT Operator

Bitwise NOT operator inverts all bits of the expression.

Syntax:

```C++
~exp
```

The following code fragment illustrates the use of the bitwise NOT operator:

```C++
~7       // evaluates to -8
```

This operator is not applicable to floating-point values.

& Bitwise AND Operator

This operator performs a bit-wise AND operation on its operands.

Syntax:

```C++
exp1 & exp2
```

Usage:

```C++
7 & 8               // evaluates to 0
2 & 2               // evaluates to 2
```

This operator is not applicable to floating-point values.

^ Bitwise XOR Operator

This operator performs a bit-wise XOR (eXclusive OR) operation on its operands.

Syntax:

```
C++
exp1 ^ exp2
```

Usage:

```
C++
7 ^ 8               // evaluates to 15
2 ^ 2               // evaluates to 0
```

This operator is not applicable to floating-point values.

| Bitwise OR Operator

This operator performs a bit-wise OR operation on its operands.

Syntax:

```
C++
exp1 | exp2
```

Usage:

```
C++
7 | 8               // evaluates to 15
2 | 4               // evaluates to 6
```

This operator is not applicable to floating-point values.

! Logical NOT Operator

Logical NOT operator inverts (logically) an expression. Evaluates to 0 if the expression is non-zero or to non-zero if the expression is zero.

Syntax:

```
C++
!exp
```

The following code fragment illustrates the use of the logical NOT operator:

```
C++
!7      // evaluates to 0
!0      // evaluates to 1
```

&& Logical AND Operator

This operator evaluates to non-zero if and only if both operands are non-zero, otherwise it evaluates to zero.

Syntax:

```
C++
exp1 && exp2
```

Usage:

```
C++
7 && 8              // evaluates to 1
0 && 2              // evaluates to 0
5 == 5 && 2==2      // evaluates to 1
```

If *exp1* evaluates to zero, *exp2* is not evaluated.

|| Logical OR Operator

This operator evaluates to non-zero if at least one of if its operands is non-zero, otherwise it evaluates to zero.

Syntax:

```
C++
exp1 || exp2
```

Usage:

```
C++
0 || 0              // evaluates to 0
0 || 2              // evaluates to 1
5 == 5 || 2==3      // evaluates to 1
```

If *exp1* evaluates to a non-zero value, *exp2* is not evaluated.

sizeof() Operator

This operator returns the size, in bytes, of the enclosed identifier.

Syntax:

```
C++
sizeof(id)
```

`id` may be either a built-in type, typedef'ed type, user-defined type or field reference. Taking the size of user-defined dynamic type is incorrect and will result in compile error.

The following code fragment illustrates the use of the `sizeof() operator`:

```
C++
struct A
{
    int a;
    int b;
    int c[4];
};

struct B
{
    int a;
    int b[a];
};

struct C
{
    B b;

    const SizeOfInt = sizeof(int);      // correct, evaluates to 4
    const SizeOfA = sizeof(A);          // correct, evaluates to 24 (size of static user-defined
type A)
    const SizeOfB = sizeof(B);          // incorrect, results in compile-time error, B is
dynamic type
    var SizeOfb = sizeof(b);          // correct, will be calculated at run-time
};
```

& Address-Of Operator

This operator takes the address (offset) of the given field.

Syntax:

```
C++
&field-id
```

Returned address is absolute.

```
C++
struct A
{
    int a;
    int b;
    const OffsetTob = &b - this;    // offset to field b (in bytes) from the beginning of the
                                    // structure is now stored in OffsetTob constant
};
```

* Multiplication Operator

This operator computes the multiplication of two given expressions.

Syntax:

```
C++
exp1 * exp2
```

Usage:

```
C++
7 * 8               // evaluates to 56
(2 + 2) * (7 - 1)    // evaluates to 24
```

/ Division Operator

This operator computes the result of division of one operand by another.

Syntax:

```
C++
exp1 / exp2
```

If the divisor evaluates to zero, a run-time error occurs. If the result of the expression is used as an integer value, the result of the division is truncated to the nearest integer value.

```
C++
4 / 2       // evaluates to 2
3 / 2       // evaluates to 1
5 / 0       // a run-time error occurs
```

% Modulo Division Operator

This operator computes the remainder of the division of the first operand by the second operand.

Syntax:

```
C++
exp1 % exp2
```

If the divisor evaluates to zero, a run-time error occurs.

```
C++
4 % 2     // evaluates to 0
5 % 2     // evaluates to 1
5 % 0     // a run-time error occurs
```

This operator is not applicable to floating-point values.

+ Addition Operator

This operator computes the sum of two given expressions.

Syntax:

```
C++
exp1 + exp2
```

Usage:

```
C++
7 + 8               // evaluates to 16
2 + 2               // evaluates to 4
```

Addition operator may also be used in string expressions to concatenate strings or to add `immediate` or `field` to a string.

- Subtraction Operator

This operator computes the difference between two given expressions.

Syntax:

```
C++
exp1 - exp2
```

Usage:

```
C++
7 - 8               // evaluates to -1
2 - 2               // evaluates to 0
```

<< Left Shift Operator

This operator performs a left shift of the first operand. The second operand specifies how many bits to shift.

Syntax:

```
C++
exp1 << exp2
```

Usage:

```
C++
7 << 1          // evaluates to 14
-3 << 3         // evaluates to -24
```

This operator is not applicable to floating-point values.

>> Right Shift Operator

This operator performs an arithmetic right shift of the first operand. The second operand specifies how many bits to shift.

Syntax:

```
C++
exp1 >> exp2
```

Usage:

```
C++
456 >> 3          // evaluates to 57
-100 >> 2         // evaluates to -25
```

This operator is not applicable to floating-point values.

>>> Right Unsigned Shift Operator

This operator performs a logical (unsigned) right shift of the first operand. The second operand specifies how many bits to shift.

Syntax:

```
C++
exp1 >>> exp2
```

Usage:

```
C++
456 >>> 3          // evaluates to 57
-100 >>> 2         // evaluates to 1073741799 (0x3ffffffe7)
```

This operator is not applicable to floating-point values.

< Less Than Operator

This operator evaluates to non-zero value if first operand is less than the second operand or to zero value otherwise.

Syntax:

```
C++
exp1 < exp2
```

Usage:

```
C++
7 < 8               // evaluates to 1
2 < 2               // evaluates to 0
```

<= Less Than or Equal Operator

This operator evaluates to non-zero value if first operand is less than or equal to the second operand or to zero value otherwise.

Syntax:

```
C++
exp1 <= exp2
```

Usage:

```
C++
7 <= 8               // evaluates to 1
2 <= 2               // evaluates to 1
3 <= 2               // evaluates to 0
```

> Greater Than Operator

This operator evaluates to non-zero value if first operand is greater than the second operand or to zero value otherwise.

Syntax:

```
C++
exp1 > exp2
```

Usage:

```
C++
8 > 7               // evaluates to 1
2 > 2               // evaluates to 0
3 > 5               // evaluates to 0
```

>= Greater Than or Equal Operator

This operator evaluates to non-zero value if first operand is greater than or equal to the second operand or to zero value otherwise.

Syntax:

```
C++
exp1 >= exp2
```

Usage:

```
C++
8 >= 7              // evaluates to 1
2 >= 2              // evaluates to 1
3 >= 5              // evaluates to 0
```

== Equality Operator

Evaluates to non-zero value if both operands are equal or to zero otherwise.

Syntax:

```
C++
exp1 == exp2
```

Usage:

```
C++
7 == 8              // evaluates to 0
2 == 2              // evaluates to 1
```

!= Inequality Operator

Evaluates to non-zero value if operands are different or to zero otherwise.

Syntax:

```
C++
exp1 != exp2
```

Usage:

```
C++
7 != 8              // evaluates to 1
2 != 2              // evaluates to 0
```

?: Conditional Operator

This is the only supported ternary operator. It evaluates to the value of its second operand if the first operand evaluates to non-zero value, otherwise, it evaluates to the value of the third operand.

Syntax:

```
C++
exp1 ? exp2 : exp3
```

Usage:

```
C++
5 == 3 ? 8 : 7              // evaluates to 7
2 == 2 ? 8 : 7              // evaluates to 8
```

Functions

Internal Functions

Built-In Functions

Device Monitoring Studio provides a number of built-in functions:

Function Name	Description
`bool(exp)`	Convert a given parameter `exp` to a boolean value. The following conversion rules apply: **Value's type** / **Behavior** `bool` — `exp` returned `int` — `false` if exp equals to zero, `true` otherwise `double` — `false` if exp equals to zero, `true` otherwise `string` — `false` if exp is an empty string, `true` otherwise `reference` — `false` if reference is not initialized, `true` otherwise Note that when a statement expects a boolean, it converts the passed value to boolean automatically. The same rules are used during this implicit conversion.
`int(exp)`	Convert a given parameter `exp` to an integer value. Floating-point numbers are truncated to nearest integer, strings are parsed as containing an integer. If parsing error occurs, the result of the conversion is zero. If passed string starts with `0`, it is considered as containing octal number. If passed string starts with `0x` or `0X`, it is considered as containing hexadecimal number. References are first dereferenced and then converted to integer.

Function Name	Description
`double(exp)`	Convert a given parameter `exp` to a floating-point value. If used on string, a string is parsed as containing a floating-point number. If parsing error occurs, the result of the conversion is zero. References are first dereferenced and then converted to floating-point.
`string(exp)`	Convert a given parameter `exp` to a string value. References are dereferenced.
`decimal(exp)`	Convert a given parameter `exp` to an integer value. Always treats a passed string as containing decimal number (regardless of any prefixes). References are dereferenced.
`hex(exp)`	Convert a given parameter `exp` to an integer value. Always treats a passed string as containing hexadecimal number (regardless of any prefixes). References are dereferenced.
`octal(exp)`	Convert a given parameter `exp` to an integer value. Always treats a passed string as containing octal number (regardless of any prefixes). References are dereferenced.
`binary(exp)`	Convert a given parameter `exp` to an integer value. Always treats a passed string as containing binary number (regardless of any prefixes). References are dereferenced.
`convert_integer(exp, radix)`	Convert a given parameter `exp` to an integer value. An integer's base is passed as second argument to the function. It is automatically converted to integer. References are dereferenced.
`format(fmt_string, ...)`	This function takes a format string `fmt_string` and arbitrary number of arguments. It returns a string after placing each passed parameter to corresponding placeholder in a format string.

Function Name	Description
`visualize(ref-expr[, flags[, encoding]])`	This function requires a reference to a field and invokes a standard `visualization algorithm` for the referenced field. Use this function to take effect of display attribute, automatic enumeration parsing and so on. Use optional `flags` value to specify rendering options: **Value** / **Type of Expression** 0 — Decimal (default) 1 — Hexadecimal
`substring(string, pos, [count])`	This function extracts a substring from a given `string`. `pos` is zero-based index of the first character of the substring and count is an optional number of characters in a substring. If omitted, substring continues until the end of the string. If `pos` is greater than the string length, an exception occurs. Count can be any positive integer or -1, which is equivalent to omitting the parameter.
`subarray(array, start_offset[, length])`	This function extracts a sub-array of a byte array. First and second arguments are required. `array` is an array from which you are extracting a sub-array and `start_offset` is a zero-based offset of the beginning of sub-array. Optional `length` parameter specifies the length of the resulting sub-array in bytes.
`element(array, position)`	Returns the byte at a given offset in an array byte array.
`length(exp)`	Return a length of a string if `exp` is a string expression, or byte array (in bytes), if `exp` is a byte array.
`find(find_where, find_what[, start_pos)`	Find a substring in a string (if both `find_where` and `find_what` expressions are strings) or one array within another, if both expressions are byte arrays. Optional `start_pos` parameter specifies the location from which to start searching. It defaults to 0. The function returns integer specifying the found location or -1 if no occurrence is found.

<table>
<tr><th>Function Name</th><th>Description</th></tr>
<tr><td><code>type(exp)</code></td><td>Determine the type of the expression exp. Returns one of the following values (symbolic names are predefined in <code>stddefs.h</code>):<table><tr><th>Symbolic Constant</th><th>Value</th><th>Type of Expression</th></tr><tr><td>BooleanType</td><td>0</td><td><code>bool</code></td></tr><tr><td>IntegerType</td><td>1</td><td><code>int</code></td></tr><tr><td>FloatingPointType</td><td>2</td><td><code>double</code></td></tr><tr><td>StringType</td><td>3</td><td><code>string</code></td></tr><tr><td>ReferenceType</td><td>4</td><td><code>reference</code></td></tr></table></td></tr>
<tr><td><code>ref(exp)</code></td><td>Take the reference of expression. Expression must be a Field Access or Array Indexing Operator. The result of this function may not be used on the left of . operator or [] operator.

A special construct <code>ref(this)</code> is used to get a reference to the current object.</td></tr>
<tr><td><code>array(exp)</code></td><td>Convert an expression to a byte array. Little-endian representation of <code>boolean</code>, integer and floating-point values are returned. If <code>exp</code> is a reference, a copy of referenced value is returned as byte array. All other conversions are prohibited.</td></tr>
<tr><td><code>is_valid(exp)</code></td><td>This function evaluates the expression and returns <code>false</code> if any exception occurs during evaluation, otherwise, it returns <code>true</code>. Expression's result is never used and is silently discarded.</td></tr>
<tr><td><code>evaluate_if(exp1, exp2)</code></td><td>This function evaluates expression <code>exp1</code> and returns its result. If any exception occurs during evaluation, it returns the value of expression <code>exp2</code>. Note that if another exception occurs during evaluation of <code>exp2</code>, it is handled as usual (propagated to user interface). This function is equivalent to

C++
<code>is_valid(exp1) ? exp1 : exp2</code>

but evaluates <code>exp1</code> only once.</td></tr>
</table>

Examples

```C++
int(2.5)           // results to 2 (integer)
int("89")          // results to 89 (integer)
double("7.8")      // results to 7.8 (floating-point)
double("string")   // results to 0.0 (floating-point)
```

Native Functions

Device Monitoring Studio supports *native functions*. A native function is a function defined in the header file that uses the same language syntax.

The following syntax is used to define a function:

```C++
optional-attributes
function f_name ( parameter-list )
{
  function-body
}

optional-attributes := [ function-attribute [, function-attribute [, …] ] ]

function-attribute := arguments_array | nooptimize

parameter-list := param-name1 [, param-name2 [, … ] ]

function-body := *(variable-declaration | statement)
```

The function may be declared at any scope (either global or user-defined). Function's name must be unique within a scope. A function can take any number of arguments (or no arguments). It is allowed to pass different number of arguments during a function call. If fewer arguments passed, the rest are undefined, if more arguments passed, extra parameters are ignored.

Only the following constructs are allowed in a function body:

- A declaration of constants or constant arrays.
- A declaration of variables or variable arrays.
- Any supported statement, including expression calculation statement.

Ability to use any statement allows you to create branches and loops. A return statement sets function's return value and exits function. A function is allowed to have multiple return statements.

A following pseudo-variables are available for a function's body:

`parameter_count`

Holds the actual number of passed parameters.

`arguments_array`

Holds values of all passed parameters. The size of the array is `parameter_count`. If function has named arguments, both named parameters and values of this array may be used to reference passed parameters.

Note: for performance reasons this array is generated only when `arguments_array` attribute is specified before the function's definition.

All parameters are always passed by value. If `arguments_array` attribute is specified and a function has named arguments, corresponding array elements and named parameters have copies of values (that is, modifying one does not modify another).

Function Scope

Each function defines its own scope. All variables declared in this scope are destroyed when function execution finishes and re-created next time it is run.

Function Optimization

Device Monitoring Studio automatically optimizes constant functions. If it encounters a function that always returns a constant value and is being used with constant arguments, it replaces the function call with a calculated value. That means that all native functions must be immutable, that is, they must not affect any global state.

If you still need to use a function that modifies a global state, use the `nooptimize` attribute to suppress function optimization.

Note that function optimization always works at the call site. Consider the following example:

```
C++
function square(x)
{
    return x * x;
}

struct A
{
    char array[square(5)];    // will be optimized to 25
    int size;
    char array2[square(size)];  // will not be optimized
};
```

External Functions

Attaching Scripts

External functions referenced in your structure definition files must be defined in separate files. Device Monitoring Studio supports functions written in JavaScript.

Use the following syntax to attach a script file to the structure definition file:

```
C++
#pragma script(path-to-script-file)
```

`Path-to-script-file` is a string containing the absolute or relative (to the structure definition file) path to a script file. Only JavaScript external files (with any extension) are supported.

As with included files, Device Monitoring Studio automatically rescans a file if it is modified outside the editor.

Examples

functions.js

```
C++
function f()
{
    return 10;
}
```

structure.h:

```
C++
#pragma script("functions.js")

public struct A
{
    char array[f()];
};
```

`javascript` Keyword

In addition, JavaScript code may be specified in the structure definition file using the `javascript` keyword:

```
C++
javascript
{
  function f()
  {
    return 10;
  }
};

public struct A
{
  char array[f()];
};
```

The number of javascript blocks is not limited. All blocks are processed before any structure is bound, so all functions declared in these blocks are always visible to any structure, regardless of the place where you define them.

External Functions

Device Monitoring Studio allows you to use external functions in expressions. External functions are defined in script files, attached to structure definition files using the #pragma script preprocessor directive.

An external function accepts zero or more parameters and must always return a value ("void" functions, or procedures are not supported). External function must be written in JavaScript.

Device Monitoring Studio automatically performs parameter type conversion when the function call is made. Although, make sure the value returned by the function is of correct type, as Device Monitoring Studio expects a value of a given type in several places, such as in array declaration:

functions.js:

```
C++
function GetArraySize()
{
    return 11/2;
}
```

structure.h:

```C++
#pragma script("functions.js")

struct A
{
    int array[GetArraySize()];         // error here: script returns a floating-point
number, integer expected
    int array[int(GetArraySize())];    // correct
};
```

Statements

Statements control the flow of execution. They are allowed on user-defined type scope and in the function body. This section describes all statements supported by the Device Monitoring Studio.

The general statement syntax may be described by the following grammar:

```C++
statement:
    statement-if | statement-switch | statement-break | statement-while | statement-for |
statement-dowhile | statement-return

statement-block:
    statement | field-declaration
```

Where `field-declaration` is either standard or user-defined type, or structure definition.

See the following topics for more information:

- if statement
- switch statement
- break statement
- while statement
- for statement
- do...while statement
- return statement

if Statement

`if` statement has the following syntax:

```
statement-if:
if (expr)
    statement-block | { statement-block * }
[else
    statement-block | { statement-block * }
]

statement-block:
    statement | field-declaration
```

`expr` is evaluated at run-time and if `true`, the first *statement-block* is evaluated, otherwise, the second *statement-block* is evaluated. If it is omitted and `expr` is false, nothing is evaluated and control flows to the next statement or declaration. If multiple statements or declarations need to be specified in the body of if or else, use the curly braces. For example:

```
C++
struct A
{
    int a;
    if (a>0)
        float b;
    else
    {
        double b;
        int c;
    }
};
```

switch Statement

`switch` statement has the following syntax:

```
statement-switch:
switch (sw-expr)
{
    case const-case-expr1:
        statement-block *
    [case const-case-expr2:
        statement-block *
    …
    ]
    [default:
        statement-block +
    ]
}
```

> **WARNING**
>
> `switch` statement's block must not be followed by a ';' character!

`switch` statement is evaluated according to the following procedure:

1. All `const-case-exprN` expressions are evaluated at compile time. If they fail to compute to a constant value, compilation error occurs. If you need to use non-const expressions, consider using the case_union.
2. `sw-expr` is evaluated at run time.
3. The resulting value is compared with each `const-case-exprN` value one by one until a match is found. If the match is not found, statement-block after the "default:" label is evaluated, if present.
4. If the match is found, all statement blocks after the corresponding "case" label are evaluated, until the "default:" label, end of switch statement or a break statement are met.

```
C++
struct A
{
    BYTE type;
    switch (type)
    {
        case 0:
            int value;
            break;
        case 1:
            double value;
            break;
        case 2:
            string value;
            break;
        default:
            $assert("Invalid file");
      }    // note: no ';' allowed here!
};
```

break Statement

`break` statement has the following syntax:

```
statement-break:
break;
```

> **NOTE**
>
> The ';' character at the end of the statement is mandatory.

The break statement may be used:

- Inside the switch statement to stop statement evaluation.
- Inside the for statement, while statement or do...while statement to cancel a loop.

In addition, `break` statement always ends the current scope, even if it is used outside of the switch or loops.

while Statement

`while` statement has the following syntax:

```
statement-while:
while (expr)
    statement-block
```

Evaluates statements and declarations in *statement-block* until `expr` becomes `false`. If expression is `false` at the first iteration, no statements are evaluated.

```
C++
var i = 10;
while (i)
{
    int a;
    i = i - 1;
}
```

for Statement

`for` statement has the following syntax:

```
statement-for:
for (init-expr; condition-expr; increment-expr)
    statement-block
```

`for` statement is equivalent to the following construct:

```
C++
init-expr;
while (condition-expr)
{
    statement-block;
    increment-expr;
}
```

It evaluates statements in *statement-block* while `condition-expr` evaluates to a non-zero value.

Note that unlike C/C++, all `for` statement expressions must be present and cannot be omitted. Device Monitoring Studio does not support unary operators like `++`, `--`, `+=`, `-=` and others, so increments must be specified in the form of `i = i + 1` instead of `++i`.

`init-expr` must have the following syntax:

```
C++
init-expr:
var name = expr
```

For example,

```
C++
for (var i = 0; i < 10; i = i + 1)
{
    int a;
}
```

do...while Statement

`do…while` statement has the following syntax:

```
statement-dowhile:
do
    statement-block
while (expr);
```

Evaluates statements and declarations in *statement-block* until `expr` becomes `false`. If expression is `false` at the first iteration, statements in statement block are evaluated exactly one time.

```
C++
var i=10;
do
{
    int a;
    i = i - 1;
} while (i);
```

return Statement

`return` statement has the following syntax:

```
return expr;
```

Evaluates the passed expression and sets the current native function's return value. It also exits the current function. If used outside the function body, an exception is generated.

```C++
function square(x)
{
    return x * x;
}
```

Scopes

A scope is a namespace for user-defined types, typedef-ed types, constants and native functions.

There is always a global scope, a scope that represents the source file itself. All enumerations, typedefs and user-defined types declared in the file (and not enclosed by other user-defined types) are said to be defined at the global scope.

```C++
// beginning of the file
typedef int MyIntType;

enum MyEnum
{
    // …
};

const MyConstant = 5;

struct MyStruct
{
    // …
};
```

In the example above, all identifiers, that is, `MyIntType`, `MyEnum`, `MyConstant` and `MyStruct` are declared in the global scope.

Each user-defined type creates its own scope:

```C++
struct A            // A is declared in the global scope and creates its own scope
{
    const B = 10;        // B is declared in scope of structure A
};
```

Every enclosed scope "sees" all its parent scopes. That is, when name-lookup is performed (for the search of the identifier), first, the most enclosed scope is searched. If the identifier is not found, a parent scope is searched and so on.

This allows an identifier to be overloaded in the enclosed scope:

```C++
const MyConstant = 10;
struct A
{
    const MyConstant = 5;
    int array[MyConstant];  // will use MyConstant from the A scope
};

struct B
{
    int array[MyConstant];  // will use MyConstant from the global scope
};
```

A compiler does not provide a way to reference `MyConstant` from the global scope from scope *A* in this

example. But once the scope is closed, a global scope is active again (see structure *B*).

Enumerations are slightly different in a way they use scopes: enumeration declaration does not create a scope but places all enumeration values into the parent scope. See the enumerations section for more information.

Constants and Constant Arrays

Constants allow you to calculate the value of an expression and store it. Compare constants with preprocessor constants. Preprocessor is run before compilation of source file occurs and therefore has limited expression evaluation capabilities.

Syntax:

```
const name-id = value-expr;
```

Constants are allowed in any scope. A `value-expr` must be constant:

```
C++
const MyConstant = 5;  // valid, constant expression

struct A
{
    int a;
    const double_a = a * 2;    // error, value-expr is not a constant
};
```

Constant Arrays

You can use the following syntax to declare an array:

```
C++
const name-id [ ] = { initializer-list };
```

Where `name-id` is a name of array and `initialize-list` is a comma-separated list of expressions that initialize array elements.

Variables and Variable Arrays

Variables work almost like constants with an exception that you can change their value at a later time.

Syntax:

```
C++
var name-id [= value-expr];
```

You do not specify the type of the variable. It is automatically determined from the type of the value-expr. You may change the variable type at a later time by assigning another value to a variable.

Variables are allowed at user-defined type scope. If `value-expression` is omitted, variable is initialized with zero. You can change the value of a variable using the following syntax:

```
C++
variable-name-id = expression;
```

Variables may be used in expressions just like constants. For example:

```C++
struct A
{
    var SomeVariable = 10;    // declare variable and assign a value to it
// …
    SomeVariable = SomeVariable * 2 - 20;  // change the value of the variable
// …
    char Data[SomeVariable];    // use variable
};
```

Optimizations

Device Monitoring Studio performs an optimization when it compiles variables. All constant expressions are evaluated to their numeric equivalents. Device Monitoring Studio can also optimize constant sub-expressions if they do not have side effects.

```C++
struct A
{
    var MyConstant = 60 * 60;    // will be optimized directly to 3600
    var PI = 3.1415926;
    var _2PI = 2 * PI;        // will be optimized directly to 6.2831852
    int a;
    var test = a * (_2PI / 180);  // sub-expression in parenthesis will be optimized
};
```

Using Variables

Variables let you overcome the limitations of standard lookup procedure. A good example is a definition of a PNG file structure (installed with Hex Editor Neo). A PNG file consists of several chunks of different type and size. Although these chunks are almost unrelated, sometimes the structure of a chunk greatly depends on some of the fields of one of the previous chunks. Using variables you may "capture" the value of such field and later use it in subsequent chunks to choose between one structure or another. See the provided sample file for more information on using variables.

Variable Arrays

You can use the following syntax to declare an array:

```C++
var name-id [ [total-elements-expr] ] [ = { initializer-list } ];
```

Where `name-id` is a name of array, `total-elements-expr` is an optional number of elements in an array. If omitted, the number of expressions in the initializer-list sets the number of elements in an array.

`initialize-list` is a comma-separated list of expressions that initialize array elements.

Enumerations

Device Monitoring Studio supports the special form of integer constants, called enumerations.

Syntax:

```
C++
// legacy syntax:
enum [name-id[<integer-type>]]
{
    value-id [ = const-expression]
    [, value-id [ = const-expression] …
    ]
};

// new syntax
enum [name-id[ : integer-type]]
{
    value-id [ = const-expression]
    [, value-id [ = const-expression] …
    ]
};
```

Enumeration may optionally have a name and a type. If name is omitted, a nameless enumeration is created. If type is omitted, it is defaulted to int.

An expression, if specified, must be a constant expression, that is, its value must be calculable at the compile-time. You may use immediates, constants and other previously defined enumerations as well as sizeof() operator in its static form.

If expression is omitted, the value of the current element is computed as a value of the previous element plus one. If this is a first enumeration element, its value will be 0:

```
C++
enum MyEnum : unsigned
{
    FirstValue,                             // defaulted to 0
    SecondValue,                            // defaulted to 1
    ThirdValue = 5,                         // overridden, equals 5
    FourthValue,                            // defaulted to 6
    FifthValue = ThirdValue - SecondValue,  // equals to 4
};
```

An enumeration does not create its own scope and places all values in the parent scope.

```
C++
enum MyEnum
{
    FirstValue            =0x00000010,
    SecondValue           =0x00000020
};

const test = FirstValue;   // valid, as FirstValue (and SecondValue) are placed in the global
scope, parent to MyEnum's scope
```

Using Enumerations

You may use named enumerations as types for fields in user-defined types. If you do this, Device Monitoring Studio will automatically recognize the enumeration value when the structure is bound to the data. For example:

```
C++
enum MyFlags : unsigned
{
    FIRST_BIT_SET       =0x00000001,
    SECOND_BIT_SET      =0x00000002,
};

struct A
{
    MyFlags flags;
};
```

When you bind structure `A` to data, *flags* gets visualized as following:

Data Value	Display	
1	FIRST_BIT_SET	
2	SECOND_BIT_SET	
0	0	
3	FIRST_BIT_SET	SECOND_BIT_SET
5	FIRST_BIT_SET	4
8	8	

Device Monitoring Studio automatically parses the enumeration value and displays its symbolic name (or names).

User-Defined Types

```
[public | private] (struct | union | protocol) [name-id]
{
    [ element-decl; [ element-decl; … ] ]
};

[public | private] case_union [name-id]
{
    case expression1:
        [ element_decl; … ]
    [
    case expression2:
        [ element_decl; … ]
    …
    ]
    [
    default:
        [ element_decl; … ]
    ]
};

element-decl:
  [ (field-decl | typedef-decl | const_decl | user-defined-type-decl) ; …]

field-decl:
  type var-decl [, var-decl …] ;

var-decl:
  (id | id[expression] | id:expression | id as type-id * [(expression)] )
```

The first syntax form allows you to define structure or union, while the second form allows you to define a case union.

When `public` keyword is placed before the structure declaration, the structure becomes a public user-defined type and appears in Protocols List Tool Window. A private keyword may be used to prevent the structure from being listed in the list of user-defined types in the Structure Binding dialog. If omitted, a structure declaration is private by default. Applies only to user-defined types, declared on the global

scope.

A special directives may appear within a declaration of a structure, union or a case union:

```
C++
hidden:
visible:
```

A `hidden:` directive hides all subsequent fields and `visible:` directive makes fields visible. By default, all fields are visible. Hiding a field only hides it from the screen, the field remain visible when referenced in expressions.

This and subsequent sections provide an in-depth description of user-defined types.

Each user-defined type creates a scope. All enclosed constants, enumerations, typedefs, native functions and user-defined types are then included into this newly created scope.

A structure, union or case union may be nameless. Nameless types are allowed anywhere besides the global scope. A nameless type may also be used in the typedef declaration:

```
C++
typedef struct
{
    // …
} A;
```

equivalent to:

```
C++
struct A
{
    // …
};
```

while the following fragment creates a structure named A and its aliases B and C:

```
C++
typedef struct A
{
    // …
} B,C;
```

Supported Types

Structures

A structure is a combination of data fields. A structure occupies space required to store all its fields, one by one, subject to *structure packing* or alignment.

A structure definition consists of zero or more of data fields:

```
type var-decl [, var-decl…];
```

where `var-decl` is:

```
(id | id[array-size-expression] | id:bit-field-size-expression | id as type-id *)
```

A structure may contain different fields with a same name. If such a field is referenced in expression, an [] operator may be used to address individual fields. If this operator is not used, the first field is referenced.

Typedefs, constants, enumerations, native functions and nested user-defined types are also allowed within a structure definition.

Plain field, array field, bit field and pointer field are described in more detail in their corresponding sections.

Empty structures are eliminated from the output. See also the noautohide attribute section.

Example:

```
C++
struct A
{
    int a;               // plain data field
    int b:3;             // bit-field
    int c[10];           // array
    int d as B *;        // pointer

    short s,t[5],u:12;   // multiple fields may be combined
};
```

Packing and Alignment

The important thing about user-defined types is the size and alignment of a type. The size of the structure is a sum of sizes of all its fields (subject to alignment). The alignment of the built-in type equals its size, and alignment of the structure is calculated by Device Monitoring Studio, taking in account the alignment of all structure fields and current structure packing value. By default, structure packing value is 1.

You may change the structure packing value using the following directive:

```
C++
#pragma pack(N)
```

where `N` is one of the following values: 1, 2, 4, 8, 16, 32.

> **NOTE**
>
> The following rule is used when computing the alignment of each structure field:
>
> Each data field starts at offset which is a multiple of its alignment. A number of unused padding bytes is inserted if required, but no more than the current structure packing value.

The implementation of structure packing and field alignment in Device Monitoring Studio complies with standard C implementation.

Byte Order

By default, Device Monitoring Studio respects the current byte order specified for the editor window. You can change the byte order at any time using the following directive:

```
C++
#pragma byte_order(LittleEndian | BigEndian | Default)
```

This directive changes the current byte order until the end of the current scope, or until another byte_order directive.

Unions

A union is a combination of data fields. In contrast to a structure, all union data fields are located at the same address and, therefore, share the same storage space. The size of the union equals to the size of the largest field, and alignment of the union equals the largest field's alignment.

A union definition consists of zero or more of data fields:

```
type var-decl [, var-decl…];
```

where `var-decl` is:

```
(id | id[array-size-expression] | id:bit-field-size-expression | id as type-id *)
```

A union may contain different fields with a same name. If such a field is referenced in expression, an [] operator may be used to address individual fields. If this operator is not used, the first field is referenced.

Typedefs, constants, enumerations, native functions and nested user-defined types are also allowed within a union definition.

Plain field, array field, bit field and pointer field are described in more detail in their corresponding sections.

Example:

C++

```
union A
{
    int intVal;
    short shortVal;
    double dblVal;
    char charArray[4];
    struct
    {
        int a;
        __int64 b;
    } nestedStruct;
};
```

Case Unions

A case union is a special construct offered by a Device Monitoring Studio to dynamically select types at run time. It is virtually impossible to describe real-world data structures using only static types (types offered by C compiler, for example). And although dynamic types greatly increase the flexibility of the language to describe data structures, they still lack the ability to select one or another type based on run time conditions.

For example, imagine we have a byte in the data structure, followed by one of three different data structures, depending on this byte's value. Device Monitoring Studio's case unions allow you to describe such data structure.

Case union syntax (copied from User-Defined Types section):

A union definition consists of zero or more of data fields:

```
C++
case_union [name-id]
{
    case expression1:
        [ element_decl; … ]
    [
    case expression2:
        [ element_decl; … ]
    …
    ]
    [
    default:
        [ element_decl; … ]
    ]
};
```

```
element-decl:
  [ (field-decl | typedef-decl | const_decl | user-defined-type-decl) ; …]

field-decl:
  type var-decl [, var-decl …] ;

var-decl:
  (id | id[expression] | id:expression | id as type-id *)
```

Case union consists of one or more *case blocks* and an optional *default block*. Device Monitoring Studio evaluates each `expressionN` and if it is nonzero, elements immediately following a case block are used. All other case blocks and default block are ignored. If none of case expressions is evaluated to a non-zero value, the *default block* is used. If the *default block* is omitted, a case union becomes an empty structure and is removed from the output.

```
C++
struct B
{
    // …
};

struct C
{
    // …
};

struct A
{
    BYTE val;
    case_union
    {
        case val == 0:
            int a;
            int b;
        case val == 1:
            B b;
        default:
            C c;
    } s;
};
```

Case Union Optimization

All case expressions are evaluated at compile time. If an expression is a non-zero constant expression, a whole case union becomes equivalent to a corresponding structure. If all case expressions evaluate to constant zero, the whole case union becomes equivalent to a structure with fields from the default block. If there is no default block in a case union, it becomes an empty structure and is eliminated from the output.

Forward Declarations

Device Monitoring Studio Neo requires that each referenced type must be defined before it is used; otherwise, the compile-time error occurs. Sometimes it is impossible or inconvenient to follow this rule. Forward declarations allow you to declare the identifier as a user-defined type, without actually defining it:

```
C++
struct B;          // forward declaration of B

struct A
{
    B b;           // compiles OK, as B has been declared
};

struct B           // actually define B
{
    // …
};
```

Syntax:

```
(struct | union | case_union) name-id ;
```

`name-id` is required in forward declarations.

All forward declarations must be resolved before the end of the source file, otherwise an error occurs. However, it is not an error not to resolve a forward declaration if it has not been referenced.

Data Fields

Plain Field

Plain field is an ordinary field of a given type.

Syntax:

```
type-id var-id;
```

Example of plain fields:

```
C++
struct B
{
    // …
};

struct A
{
    int a;
    B b;
};
```

Both `a` and `b` fields of structure `A` are plain fields.

Array Field

Array field describes an array of some data type. Array is a sequence of several values of the same type.

Syntax:

```
[[noindex]] type-id var-id[expression];
```

`Expression` may be constant expression or dynamic expression. If it evaluates to 0, the field is removed from the output.

Device Monitoring Studio automatically distinguishes between a simple array and an ordinary array. An array of elements of integer and floating-point built-in types is considered a simple array. Simple arrays do not impose a limit on a number of items and are extremely cheap to bind and consume no memory at all. Ordinary arrays (that is, arrays of user-defined types, or string types) have a hard-coded limit on array item count (about 2 million items) and require a corresponding amount of free RAM. A `noindex attribute` may precede the array field declaration if you do not use an Array Indexing Operator in any of the expressions to refer to elements of this array.

This will save the memory and structure binding time.

Example of array fields:

```
C++
struct B
{
    // …
};

struct A
{
    int a[5];      // static array, considered as simple array
    B b[a[0]];     // dynamic array (a number of elements is taken from the first element of a).
Not a simple array
};
```

Simple and Ordinary Arrays

Any array of elements of integer or floating-point built-in types is called a simple array. Array of elements of other types is called an ordinary array.

The following table briefly describes the difference between two array types:

Property	Simple Array	Ordinary Array
User-defined element type supported	No	Yes
Consumes more memory as array's size increases	No	Yes
Consumes more time as array's size increases	No	Yes
Has upper array size limit	No	Yes
Supports [noindex] attribute	No, ignored if used	Yes, consumes less memory and time if used on an array

"Infinite" Arrays

You are allowed to declare an array of "infinite" size when declaring an array. The following syntax is used:

```
type-id var-id[*];
```

`type-id` must be a user-defined type that must contain at least one $break_array directive that will specify the last element of the "infinite" array.

This is very useful if array size is not known at the array declaration point. For example, the following code snippet is capable of parsing a C-style null-terminated string:

```
C++
struct StringCharacter
{
    char c;
    if (c == 0)
      $break_array(true);
};

struct NullTerminatedString
{
    StringCharacter chars[*];
};

// Display the class usage
struct FileStructure
{
    NullTerminatedString FileName;
    NullTerminatedString Location;
    // …
};
```

Visualization

When Device Monitoring Studio visualizes an array, it optionally visualizes its first several values in-line. Other values are displayed when you expand the array item. Character arrays (arrays of elements of char and wchar_t types) are visualized as ANSI or UNICODE strings respectively.

Bit Field

Bit field is an integer field which occupies less space than the underlying integer type.

Syntax:

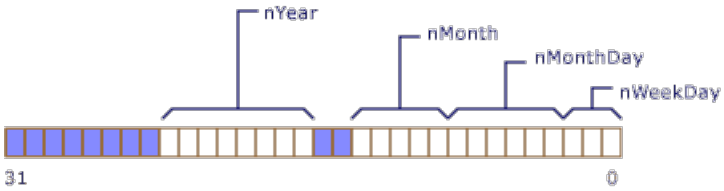
```
typeid var-id:expression;
```

The underlying `type-id` of a bit field must be an integer type. `Expression` may be constant expression or dynamic expression.

Example of bit fields:

```
C++
struct Date
{
    unsigned short nWeekDay  : 3;    // 0..7   (3 bits)
    unsigned short nMonthDay : 6;    // 0..31  (6 bits)
    unsigned short nMonth    : 5;    // 0..12  (5 bits)
    unsigned short nYear     : 8;    // 0..100 (8 bits)
};
```

The conceptual memory layout of an object of type `Date` is shown in the following figure.



Note that `nYear` is 8 bits long and would overflow the word boundary of the declared type, unsigned short. Therefore, it is begun at the beginning of a new unsigned short. It is not necessary that all bit fields fit in one object of the underlying type; new units of storage are allocated, according to the number of bits requested in the declaration.

The ordering of data declared as bit fields is from low to high bit, as shown in the figure above.

Pointer Field

Pointer field is functionally equivalent to a plain field but additionally describes the type this field points to. When Device Monitoring Studio binds a pointer field, it automatically binds a pointed type at a calculated offset.

Syntax:

```
type-id var-id as pointed-type-id * [(expression)];
```

`Expression`, if present, is evaluated at run-time and the result is added to a field value. `type-id` must be an integer type. `pointed-type-id` must be a user-defined type.

Example of pointer field:

```
C++
struct B
{
    // …
};

struct A
{
    short ptr1 as B *;
    unsigned int ptr2 as B *(10);     // B will be bound at offset (ptr2 + 10)
};
```

The resulting offset must be an absolute offset in a file. In cases where fields contain only relative offsets, this keyword may be used in an expression to “convert” a relative offset to an absolute offset:

```
C++
struct B;

struct A
{
    short ptr1 as B *(this);  // B will be bound at offset (this + ptr1)
};
```

When pointers are processed and pointed structures are bound at resulting offsets, the current structure scope is used as a parent scope for a bound structure. This allows referencing its fields from the pointed structure:

```
C++
struct B;

struct A
{
    int a;
    int ptr as B *;
};

struct B
{
    int array[a];     // will reference a in A, if B is automatically bound via a pointer
};
```

Late-evaluation is performed for pointers, which allows pointed types to reference fields from the parent scope even if they are defined below the pointer field.

Attributes

Field Attributes

Device Monitoring Studio supports a number of useful attributes that change the default behavior for individual bound fields. You should put attributes before a field you want them affect. The following syntax is supported:

```
attribute-list:
[ *attribute-decl ]

attribute-decl:
noindex | noautohide | read(expr) |
format(expr) | description(expr) | color_scheme(expr)
```

See the following sections for attribute descriptions.

Type Attributes

In addition to field attributes, a single display attribute is supported on types. It allows the user to change the default visualization for a type.

Field Attributes

noindex Attribute

Syntax:

```
noindex
```

Specifying this attribute for an array field turns off automatic building of array index. Array index is required for Array Indexing Operator to work properly. This attribute is ignored if used on simple array or non-array field.

noautohide Attribute

Syntax:

```
noautohide
```

By default, Device Monitoring Studio eliminates fields which have zero size during binding. Specifying this attribute allows you to turn this behavior off.

onread Attribute

Syntax:

```
onread(expr)
```

Allows you to change the way this field is read by the Device Monitoring Studio. You may specify expression to be evaluated each time Structure Viewer accesses the field's value.

Take the following notes into consideration:

- Your expression will be evaluated each time the field's value is read from the document, even during binding.
- Device Monitoring Studio caches field values. This means that the result of your expression must be persistent. That is, for any `x` and `y`, if `x == y`, `expr(x) == expr(y)`.

In expression you may refer to actual field's value using a special variable `_1`:

```
C++
struct A
{
    // The following field stores the size of the array minus 2
    [onread(_1 + 2)]
    int array_size;   // will be displayed as actual value + 2
    char array[array_size];
};
```

format Attribute

Syntax:

```
format(const-string-expr)
```

Use this attribute to set the format string used during visualizing of the field's value. Note that only part of the full format string must be specified: instead of "{0b16}" use just "b16". `const-string-expr` is evaluated at compile time.

description Attribute

Syntax:

```
description(const-string-expr)
```

Set the field description to be displayed in Device Monitoring Studio user interface. `const-string-expr` is evaluated at compile time.

color_scheme Attribute

Syntax:

```
color_scheme(const-string-expr)
```

Set the color scheme to be used by Device Monitoring Studio to visualize the field. `const-string-expr` is

evaluated at compile time.

Type Attributes

display Attribute

Syntax:

```
display(expr)
```

This attribute may be used to override the default type's visualization algorithm.

When Device Monitoring Studio generates a value for a collapsed type, it by default displays values of first 5 type's fields. This attribute allows you to change that.

Device Monitoring Studio evaluates the `expr` expression in the context of the current type (that is, you may reference all type fields directly). The result is then converted to string and displayed in Structure View data visualizer.

The following example renders the user-friendly MAC address:

```
C++
[display(format("{0b16Xw2arf0}:{1b16Xw2arf0}:{2b16Xw2arf0}:{3b16Xw2arf0}:{4b16Xw2arf0}:
{5b16Xw2arf0}",
  data[0],data[1],data[2],data[3],data[4],data[5]))]
struct MAC
{
    unsigned char data[6];
};
```

The following example displays the sum of array's values:

```
C++
function sum(r)
{
    var sum = r[0];
    for (var i = 1; i < 6; i = i + 1)
        sum = sum + r[i];
    return sum;
}

[display(sum(ref(data)))]
struct MAC
{
    unsigned char data[6];
};
```

Typedefs

You can create an alias for any built-in or user-defined type.

Syntax:

```
C++
typedef existing-type new-type-id [,new-type-id …];
```

Type alias definitions are allowed at any scope. `existing-type` must be either built-in type, enumeration or user-defined type. Nameless types are allowed.

```C++
typedef int INT;
typedef struct { int a,b; } MyStruct;

struct A
{
    // …
};

typedef A B,C,D;
```

Typedef does not create a new type, it only creates an alias for an existing type. You may reference a type by its original name, or by one of its aliases.

Directives

Directives are special commands to the compiler which are only allowed at non-global scope.

View the subsequent topics to get detailed description of each supported directive.

$assert Directive

Syntax:

```C++
$assert(condition-expression [, message [, fatal-expression]]);
```

Assert directive is evaluated at run-time. First, `condition-expression` is evaluated. If it evaluates to non-zero value, nothing happens. But if it evaluates to a zero value, message is displayed to the user and structure binding is terminated. If message is omitted, standard "Assertion Failed" message is displayed. `message`, if present, must be a string expression. `fatal-expression`, if present, must be a constant expression. If it evaluates to a non-zero value, the assertion is fatal (this is a default behavior), that is, fired assertion terminates structure binding. If it evaluates to zero, assertion is only informational.

Assertions are good at verifying whether the data structure is being bound to correct data.

```C++
struct A
{
    int a;
    $assert(5 < a && a < 10,"a must be between 5 and 10");
};
```

$print Directive

Syntax:

```C++
$print(var-name-string-expression, var-value-expression);
```

The first expression must be a constant string expression and serves as a pseudo-field name. This name is displayed in the Structure View data visualizer. It is also added as a real field into the current scope and may be later referenced in expressions. The second expression is a field's value. It may be either a constant expression or a non-const expression, that will be evaluated at run time.

```C++
struct A
{
    int a;
    int b;
    $print("double_a", a * 2);
    $print("a/b", double(a) / b);
};
```

Both `$print` directives in the example above introduce new fields into the current scope (of struct `A`), but only the first one may be referenced in expressions, because the second one has incorrect name. So, it's OK to use any name for the first argument, but only syntactically correct ones may be referenced in expressions.

$break_array Directive

Syntax:

```C++
$break_array(const-conditional-expression);
```

When used in a user-defined type, unconditionally breaks an enclosed array.

The parameter, which must be a constant value, specifies whether the current element should be included into an array (`const-conditional-expression` = true), or not (`const-conditional-expression` = false)

May be used either in conjunction with infinite array or with an ordinary array. If used not inside an array, the directive is ignored.

```C++
struct StringCharacter
{
    char c;
    if (c == 0)
      $break_array(true);
};

struct NullTerminatedString
{
    StringCharacter chars[*];
};
```

$bind Directive

Syntax:

```C++
$bind(type-string-expr, var-string-expr, addr-expr);
```

All expressions are evaluated at run time. First two are automatically converted to strings, while the third is expected to be of integer type.

This directive instructs parser to bind another structure to a given address. Binding is delayed until the current structure binding is successfully finished.

> **NOTE**
>
> You must reference the full type name, for example `struct MyStruct`, not the `MyStruct`, and the referenced type must have been declared as public.

```C++
public struct B
{
// …
};

public struct A
{
    int Offset;
    if (Offset != 0)
        $bind("struct B","pB",Offset);
};
```

$alert Directive

Syntax:

```C++
$alert(expr);
```

Evaluates `expr` at bind time and displays the result in a message box.

$revert_to Directive

Syntax:

```C++
$revert_to(reference-expression);
```

This directive updates the `current_offset`. It may be used to have look ahead in a structure. `reference-expression` is evaluated at run-time and must be a reference to a field in the current type. After this directive executes, `current_offset` becomes the start of a field.

```C++
struct A
{
    int type;
    switch (type)
    {
        case 0:
            B b;
            break;
        case 1:
            C c;
            break;
        default:
            $revert_to(ref(type));
            D d;
    }
};
```

$shift_by Directive

Syntax:

```C++
$shift_by(integer-expression);
```

This directive updates the `current_offset`. It may be used to have look ahead in a structure. `integer-expression` is evaluated at run-time and must be an integer value, which is added to

`current_offset` upon directive execution.

```
C++
struct A
{
    int skip_bytes;
    $shift_by(skip_bytes);
    int next_field;
};
```

$remove_to Directive

Syntax:

```
C++
$remove_to(reference-expression);
```

This directive removes one or more last bound fields until the referenced field. `reference-expression` is evaluated at run-time and must be a reference to a field in the current type.

> **WARNING**
>
> All fields being removed must be visible and no hidden fields must be between them.

```
C++
struct A
{
    int type;
    switch (type)
    {
        case 0:
            B b;
            break;
        case 1:
            C c;
            break;
        default:
            $remove_to(ref(type));
            D d;
    }
};
```

Format String Syntax

This section describes the format string syntax. Format string is used in format() function and format attribute.

Library format string syntax is not compatible with the standard printf syntax. Instead, it has a different syntax.

The format string has blocks of plain text which are directly copied to the output and parameter placeholders. Each placeholder has the following syntax:

```
{<param-index>[width-decl][alignment-decl][plus-decl]
[precision-decl][base-decl][padding-decl][ellipsis-decl]
[locale-decl]}
```

The placeholder must be enclosed in curly braces. If you need to use the opening curly brace in the text, you need to duplicate it to distinguish from the placeholder beginning. There is no need to escape the closing brace, it will always be parsed correctly.

Parameter declaration starts with a parameter's number. This is the only mandatory field. Parameters are ordered starting from zero. All subsequent declarations are optional. If several declarations are used, their order is not significant and there must be no space or any other separator between them.

`width-decl`

Use this declaration to limit the minimum and/or maximum length of a rendered parameter, in characters. The syntax of a declaration is one of:

- `w<min-width>,<max-width>`
- `w<min-width>`
- `w,<max-width>`

Both `min-width` and `max-width` must be decimal integers and if specified, `max-width` must be larger than `min-width`.

`alignment-decl`

Use this declaration to set parameter alignment. It is ignored unless `width-decl` is also used. Use one of:

- `al` – align left (default)
- `ar` – align right
- `ac` – align center

`plus-decl`

Forces the plus sign to be rendered for positive numbers. Syntax:

- `+`

`precision-decl`

Use the declaration to specify the number of digits to be displayed after the comma. Used only for floating-point types. If not specified, the default one (6) is used.

- `p<number>`

`base-decl`

Specify a base for an integer. If any base besides 10 is used with the floating-point type, only the integer part is rendered. Only bases of 2, 8, 10, and 16 are supported. Lowercase or uppercase hexadecimal may be specified:

- `b2` – binary
- `b8` – octal
- `b10` – decimal (default)
- `b[0]16[x]` – lowercase hexadecimal. If prefix "0" is used, library adds "0x" before the number
- `b[0]16X` – uppercase hexadecimal. If prefix "0" is used, library adds "0X" before the number

`padding-decl`

Set the character to fill the space when `min-width` is set (see the `width-decl` above). The default one is space.

- `f<character>`

`ellipsis-decl`

Add the ellipsis sign when truncating output. It is not compatible with center alignment (will act as left alignment).

- `e`

`locale-decl`

Separate thousands with the default user locale's thousand separator. Will work only for base 10.

- `l`

Errors

This topic describes all compilation and binding errors generated by Device Monitoring Studio.

CE001: The requested operation is not allowed on the given data type or not expected here

Operation or operator you attempted to use is not supported. Example:

```C++
var result = "string" - 2;  // operator - is not supported for strings
```

CE002: Divison by zero

Divison by zero has been encountered.

```C++
int array[5/0];
```

CE003: The specified identifier was not found

You referenced the previously undeclared identifier.

```C++
public struct A
{
    int a;
    int b;
    int arr[c];   // generates CE003, c is undeclared
};
```

CE004: Scalar is expected

A requested operation is allowed only on scalar values.

```C++
public struct A
{
    int data[10];
    int array[data];  // generates CE004, data is not a scalar
};
```

CE005: Vector is expected

A requested operation is allowed only on vector values.

```C++
public struct A
{
    int a;
    int b[a[1]];  // generates CE005, a is not a vector
};
```

CE006: Array index is out of range

An attempt to access array's element that is outside of the declared array size.

```C++
public struct A
{
    int a[10];
    int b[a[12]]; // generates CE006, a only has 10 elements (0..9)
};
```

CE007: Not implemented yet

This operation has not yet been implemented.

CE008: Invalid bit field size

Unsupported bit field size is used.

CE009: Syntax error

See error's additional message for detailed syntax error information.

CE010: Type has only been forward-declared

An attempt to materialize a type that has only been forward-declared is detected.

```C++
// forward declare B
struct B;

public struct A
{
    B data;    // generates CE010
};

// end of file
```

CE011: Operation not supported for dynamic type

sizeof operator is used with dynamic type.

```C++
struct B
{
    int size;
    char data[size];
};

public struct A
{
    char reserved[sizeof(B)]; // generates CE011, B is dynamic type
};
```

CE012: Type is redefined

An attempt to redefine already defined type is detected.

```C++
struct B
{
    int a;
};

struct B  // generates CE012
{
    int c;
};
```

CE013: Assertion failed

Assertion (generated by $assert directive) has failed.

CE014: Constant expression is expected

Compiler expects a constant expression here.

CE015: Constant string expression is expected

Compiler expects a constant string expression here.

CE016: Wrong number of arguments for a function call

An invalid number of arguments used in a call to built-in function.

CE017: Subscript operation for non-indexed array (hint: remove [noindex] attribute)

[] operator has been used for non-indexed array. Remove the noindex attribute.

```cpp
struct B { … };

struct A
{
    [noindex] B data[1000];

    char reserved[data[5].size];  // generates CE017, remove [noindex] from previous line
};
```

CE018: Error returned by JavaScript engine

JavaScript function returned an error.

CE019: Invalid argument

Invalid argument has been passed to built-in function.

CE020: Maximum allowed recursion depth is reached

Check your source file for infinite recursion.

```cpp
struct B;

struct A
{
    B b;
};

struct B
{
    A a;
};

// will generate CE020 after several iterations
```

CE021: Expression of another type is expected

Compiler expects expression of another type here. Additional information specifies what type is expected.

Scripting

Device Monitoring Studio's scripting module exposes programming interfaces for a number of application internal objects. The current implementation exposes interfaces for the following objects:

- Scripting Site Object
- Monitoring Object
- Serial Terminal Object
- MODBUS Builder Object
- Network Manager Object
- Remote Connection Manager Object
- Bridge Manager Object
- File Manager Object
- HID Manager Object

Future releases of the Device Monitoring Studio (Serial, USB, Network Sniffer) will extend the list of scriptable objects.

Scripting System Changes

Version 8.03 adds HID Manager Object.

Version 8.02 updates the scripting support:

Script Debugger

Script debugger provides step-by-step execution, breakpoints, evaluation of variables and viewing of stack trace.

File Manager Object

File Manager Object provides access to a file system.

Network Manager Object

Network Manager Object encapsulates TCP session, UDP session and TCP Listener objects.

Reworked Serial Terminal

Serial Terminal Object has been updated with breaking changes.

Reworked MODBUS Scripting API

MODBUS Send Object has been removed and new MODBUS Builder Object has been introduced.

Version 7.81 updates the scripting support:

Updated TypeScript

TypeScript has been updated to version 2.3.

JavaScript ES2017 support

The included compiler now supports JavaScript ES2017, including `async`/`await`.

Network Manager Object (breaking change)

TCP Manager Object has been renamed to Network Manager Object.

UDP session

Added support for UDP Session.

`delay` **function**

A new global function delay has been added.

Version 7.70 updates the scripting support:

TypeScript support

TypeScript is now always enabled.

JavaScript ES6 support

JavaScript ES6 is now supported by default.

JavaScript typed arrays are now used in API

Various API functions and events have been updated to use typed arrays.

Better performance

Scripting system has increased performance.

Scripting support in version 7.13 has changed significantly. This section lists all changes and also marks breaking changes:

Scripting language support (breaking change)

Only JavaScript is supported. Support for other scripting languages (like VBScript) has been discontinued.

Naming convention (breaking change)

New method naming scheme: camelCase.

New event system (breaking change)

Device Monitoring Studio does not use Automation-compatible event system anymore. Instead,

user script may directly bind JavaScript functions as event handlers. This simplifies event binding and consuming.

Simplification of many methods (breaking change)

Many MODBUS and Serial Terminal methods were simplified. For example, methods that took a variable number of arguments now receive a single JavaScript array argument.

Updated Scripting Site object (breaking change)

Several methods of Scripting Site object have been updated and a few new ones added.

New built-in object: Monitoring

New monitoring object provides methods to create, configure and manage monitoring sessions.

(Optional) support for TypeScript

User scripts may now be written in TypeScript (superset of JavaScript) for better syntax and strong type-checking. If supported by user operating system, user scripts are always checked against a TypeScript interface definition providing better error reporting and syntax checking. TypeScript compiles to plain JavaScript which is then executed.

Scripting in User Interface

Scripting Tool Window has been removed in Device Monitoring Studio 7.13. Now user scripts are opened in their own client windows inside application frame.

Working with Scripts

To create new empty script file, execute the **File » New » Script** command. To open an existing script file, execute the **File » Open...** command.

Script files are opened in their own windows. You may use the built-in editor to modify script text.

Running Scripts

To start script execution, execute the **Tools » Run Script** command. TypeScript compiler is invoked first to compile the user script. During compilation, an extensive syntax error detection is performed. If any errors are found, they are displayed in script output and highlighted in the script text.

After script is checked and compiled, the execution of its global scope starts. It continues until the last statement of the global scope finishes execution. Script is then halted but not stopped completely. This is done in order for script to be able to process scheduled asynchronous functions or bound events. Note that scheduled asynchronous functions and bound event handlers cannot be executed until the global script scope execution is finished. The exception is event handlers invoked during execution of the main script body.

To stop script execution, use the **Tools » Stop Script** command.

Persistence

There are a number of options that govern the behavior of automatic script persistence. All these options are configured in the Tools » Settings, General Tab.

- “Reload scripts on startup” option loads all named scripts each time application is started.
- “Ask to save unsaved scripts on close” option pops up a message box asking you to save all modified scripts on application close.
- “Include scripting configuration into workspace” option includes all opened scripts into the workspace.

Command Line Support

Full paths to script files passed in command line to Device Monitoring Studio are automatically loaded and opened. If `-run` option prepends a path, corresponding script file is automatically executed after being opened.

Debugging Scripts

To start script debugging put at least one breakpoint and execute the **Debug » Debug Script** command.

Placing Breakpoints

To place a breakpoint use a mouse (click on the leftmost area next to a line you want to put breakpoint on) or move the cursor to the line and execute the **Debug » Toggle Breakpoint** command.

Other commands in the Debug menu allow you to remove all breakpoints or to break a running script asynchronously.

Whenever a breakpoint is hit, script execution enters the break state.

Break State

While in break state, **Debug Watch Tool Window** and **Debug Stack Trace Tool Window** display the current execution state. **Debug Watch** window automatically shows you all local and global variables and allows entering new variable names or expressions to execute and display. **Debug Stack Trace** window shows you the current stack frame and all calling frames. You can switch to any frame by double-clicking it in the list to change the context of the Debug Watch window.

Stepping through the Code

While in break state, use the following commands to step through the code:

Step In

This command executes the current statement and then enters the break state again. If the current statement is a function call, execution “enters” the function and stops at the function beginning.

Step Over

This command executes the current statement and then enters the break state again. If the current statement is a function call, the function body is executed and execution stops again when the function returns.

Step Out

Continues execution until the currently executed function returns to its caller.

Continue

Continues the execution until the next breakpoint is hit or script execution finishes.

What Can I do with Scripting?

In current implementation, Scripting module provides access to the following objects:

- Scripting Site Object
- Monitoring Object
- Serial Terminal Object
- MODBUS Builder Object
- Network Manager Object

- Remote Connection Manager Object
- Bridge Manager Object
- File Manager Object
- HID Manager Object

Event Binding

Some of the objects available for scripting provide one or more events. For example, Serial Terminal Session Object exposes two events: ITerminalSession.sent and ITerminalSession.received. Remote Connection Manager provides the IRemoteHost.connected and IRemoteHost.disconnected events.

To bind a function to event, call the first function overload passing the function object that matches the required signature. This function returns an event id that user script may pass to the second event overload to unbind the event handler. It is not mandatory to unbind the handler before the target object is destroyed.

Once bound, the passed function object is executed each time the object event fires.

When the last statement of the global scope finishes execution, script execution is halted, but not stopped completely. This is done in order for script to be able to process scheduled asynchronous functions, bound events and promises continuations.

```
TypeScript
// Bind function to IRemoteHost.connected event
var bound_event_id = remote.connected(name => alert("Connected to " + name));

// Connect to remote computer
remote.connectServer("servername");

// Successful connection will cause our event handler to be called
// ...

// Optionally, disconnect our handler
remote.connected(bound_event_id);
```

Scripting Site Object

Scripting Site object provides a way for a script to display text in script console window (see alert function for more information). It also allows a script to query a user for some text, using the input function.

Event binding has changed in version 7.13. Now objects expose events directly. User script may bind an anonymous JavaScript function to be called when event is fired.

Scripting site object also provides the ability for delayed execution using async function. cancelAsync function may be used to cancel delayed execution. loadTextFile function can be used to load the contents of a text file into a string variable.

delay function returns a promise object that gets completed in a given number of milliseconds.

The Scripting Site object is “virtual”. Its methods are declared in global scope.

IScriptingSite Interface

```typescript
interface IScriptingSite {

    // Methods
    alert(message: string): void;
    input(message: string): string;
    async(handler: () => void, ms: number, repetitive?: boolean): number;
    cancelAsync(handlerId: number): void;
    loadTextFile(path: string): string;
    delay(ms: number): Promise<void>;
}
```

```csharp
// This interface is not available in managed environment
```

```cpp
// This interface is not available in native environment
```

IScriptingSite Methods

---

alert

```typescript
alert(message: string): void;
```

```csharp
// This method is not available in managed environment
```

```cpp
// This method is not available in native environment
```

Parameters

`message`

The string to print in the script console. Add the '\n' character to the string to insert a new line.

Description

Prints the given string to the script console.

Example

```typescript
for (var i = 0; i < 100; i++)
    alert("i equals to " + i + "\n");
```

---

input

```typescript
input(message: string): string;
```

```csharp
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`message`

The message to be displayed to the user.

Return Value

The string entered by the user.

Description

Displays a message to the user and asks him to enter the line of text. The method then returns the text returned by the user. The call to this method results in a message box to be displayed.

Example

```
TypeScript
var name = input("Enter your name:");
alert("Hello, " + name);
```

---

async

```
TypeScript
async(handler: () => void, ms: number, repetitive?: boolean): number;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`handler`

JavaScript function that takes no parameters and returns nothing. This function is invoked after `ms` milliseconds once or until cancelled, depending on the `repetitive` parameter.

`ms`

A number of milliseconds to wait until calling the passed function.

`repetitive`

An optional boolean that tells if async handler should be called once (repetitive is omitted or equals to false) or until cancelled (repetitive equals to true).

Return Value

Returns an asynchronous function identifier. You may pass this identifier to IScriptingSite.cancelAsync method to cancel delayed function.

Description

Schedules a passed Javascript function for delayed execution. A caller may optionally specify if the async function should be repetitive.

Example

TypeScript lambda that is executed once after 1 second:

```TypeScript
async(() => alert("Async handler executed"), 1000);
```

JavaScript function that is invoked every 2 seconds until cancelled after 20 seconds:

```JavaScript
var h1 = async(function() { alert("Async handler executed"); }, 2000, true);
async(function() { cancelAsync(h1); }, 20000);
```

---

cancelAsync

```TypeScript
cancelAsync(handlerId: number): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`handlerId`

Async handler identifier to cancel.

Description

Cancels pending async handler.

Example

JavaScript function that is invoked every 2 seconds until cancelled after 20 seconds

```JavaScript
var h1 = async(function() { alert("Async handler executed"); }, 2000, true);
async(function() { cancelAsync(h1); }, 20000);
```

---

loadTextFile

```TypeScript
loadTextFile(path: string): string;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`path`

Full path to a text file.

Description

Loads contents of a text file into a string.

Example

Load text file into string and print it:

```JavaScript
alert(loadTextFile("c:\\temp\\test.js"));
```

---

delay

```TypeScript
delay(ms: number): Promise<void>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`ms`

A number of milliseconds to wait until completing the returned promise object.

Description

Returns a promise that gets completed in a given number of milliseconds.

Example

Using await:

```TypeScript
async function test() {
    // ...
    await delay(500);
    // ...
}
```

Using continuations:

```TypeScript
delay(500).then(() => { ... });
```

Monitoring Object

Monitoring object provides monitoring session management. It is exposed by the global object `monitoring` and implements the IHost interface.

User script may query the list of all running monitoring sessions with IHost.sessions property or create

new monitoring session using IHost.createSession method.

Automatic Generation of Session Configuration Script

Device Monitoring Studio support automatic generation of script code that creates, configures and starts a monitoring session. To get this automatically generated script, open the Session Configuration Window, add Devices, add and configure **Data Visualizers**, configure Capture Filter and press the **Generate Script** button.

IHost Interface

```TypeScript
interface IHost {
    // Properties
    sessions: ISession[];

    // Methods
    createSession(deviceName?: string, serverName?: string): ISession;
    createSession(device?: { serial: string }, serverName?: string): ISession;
    createSession(device?: { usb: { device?: string; port?: number; address?: number; } },
serverName?: string): ISession;
    createSession(device?: { network: string }, serverName?: string): ISession;
    createSession(device?: { virtual: any }, serverName?: string): ISession;
    createSession(device?: { playback: { path: string; stream?: number; } }, serverName?:
string): ISession;
    createSession(device?: { bridge: string | IBridge }, serverName?: string): ISession;
}
```

```C#
// This interface is not available in managed environment
```

```C++
// This interface is not available in native environment
```

IHost Properties

---

sessions

```TypeScript
sessions: ISession[];
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Returns the array of session objects.

Example

Obtain a list of monitoring sessions and print their states:

```JavaScript
var sessions = monitoring.sessions;
for (var i = 0; i < sessions.length; ++i)
    alert(sessions[i].state);
```

IHost Methods

---

createSession

```TypeScript
createSession(deviceName?: string, serverName?: string): ISession;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`deviceName`

An optional string which is parsed by each installed data source in turn until the matched device is found. If omitted, device may later added using ISession.addDevice method.

`serverName`

Optional remote server name. If omitted, or passed empty string or ".", local monitoring session is created.

Description

Creates an unconfigured monitoring session. You must finish configuring the monitoring session before starting it with a call to start it. This method may be passed the same arguments as in ISession.addDevice method.

---

createSession

```TypeScript
createSession(device?: { serial: string }, serverName?: string): ISession;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`device`

A serial device identifier (full name or COM port).

`serverName`

Optional remote server name. If omitted, or passed empty string or ".", local monitoring session is

created.

Description

Creates an unconfigured monitoring session. You must finish configuring the monitoring session before starting it with a call to start it. This method may be passed the same arguments as in ISession.addDevice method.

---

createSession

```TypeScript
createSession(device?: { usb: { device?: string; port?: number; address?: number; } },
serverName?: string): ISession;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`device`

An USB device (located either by full device name, port or address, or any combination).

`serverName`

Optional remote server name. If omitted, or passed empty string or ".", local monitoring session is created.

Description

Creates an unconfigured monitoring session. You must finish configuring the monitoring session before starting it with a call to start it. This method may be passed the same arguments as in ISession.addDevice method.

---

createSession

```TypeScript
createSession(device?: { network: string }, serverName?: string): ISession;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`device`

A network device identifier (full adapter name).

`serverName`

Optional remote server name. If omitted, or passed empty string or ".", local monitoring session is

created.

Description

Creates an unconfigured monitoring session. You must finish configuring the monitoring session before starting it with a call to start it. This method may be passed the same arguments as in ISession.addDevice method.

---

createSession

```TypeScript
createSession(device?: { virtual: any }, serverName?: string): ISession;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`device`

An object of the following type: to reference Import Data Source.

`serverName`

Optional remote server name. If omitted, or passed empty string or ".", local monitoring session is created.

Description

Creates an unconfigured monitoring session. You must finish configuring the monitoring session before starting it with a call to start it. This method may be passed the same arguments as in ISession.addDevice method.

---

createSession

```TypeScript
createSession(device?: { playback: { path: string; stream?: number; } }, serverName?: string):
ISession;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`device`

An object which references a given log file and optionally a stream in it.

`serverName`

Optional remote server name. If omitted, or passed empty string or ".", local monitoring session is

created.

Description

Creates an unconfigured monitoring session. You must finish configuring the monitoring session before starting it with a call to start it. This method may be passed the same arguments as in ISession.addDevice method.

---

createSession

```TypeScript
createSession(device?: { bridge: string | IBridge }, serverName?: string): ISession;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`device`

A serial bridge name or a reference to IBridge interface.

`serverName`

Optional remote server name. If omitted, or passed empty string or ".", local monitoring session is created.

Description

Creates an unconfigured monitoring session. You must finish configuring the monitoring session before starting it with a call to start it. This method may be passed the same arguments as in ISession.addDevice method.

Example

Create new (unconfigured) monitoring session:

```javascript
// Create new monitoring session (without adding device):
var session = monitoring.createSession();

// Create new monitoring session and add COM1 serial device to it (use automatic matching):
var session = monitoring.createSession("COM1");

// Create new monitoring session and add specific serial device:
var session = monitoring.createSession({ serial: "Communications Port (COM1)" });

// Create new monitoring session and add specific USB device:
var session = monitoring.createSession({ usb: { device: "USB Input Device", port: 4, address:
1 } });

// Create new monitoring session and add specific network adapter:
var session = monitoring.createSession({ network: "Realtek PCIe GBE Family Controller(802.3)"
});

// Create new monitoring session and add specific log file:
var session = monitoring.createSession({ playback: { path: "C:\\temp\\Communications Port
(COM1)$140408$1.dmslog8", stream: 0 } });

// Create remote monitoring session for serial port
var session = monitoring.createSession("COM5", "remote_server_name");
```

Monitoring Session Object

Monitoring session object represents a monitoring session. It exposes the ISession interface. User script gets a reference to the session object either through IHost.sessions property or by calling the IHost.createSession method.

To query the current session state, check the ISession.state property.

Adding Devices and Configuring Session

Before monitoring session can be started, at least one device must be added to the session. A device may be added directly during a call to IHost.createSession method, or later using ISession.addDevice method. After the first device is added to the session, a session gets its Data Source. Data source may not be subsequently changed, therefore, if user script needs to add another device to the session, it must add device of the same type (that is, serial device to serial session, USB device to USB session and so on).

If assigned Data Source supports (or requires) configuration, it can be configured using ISession.configureSource method. Session's Capture Filter may be configured by calling ISession.setCaptureFilter method.

Devices of any installed data source, except Remote Source, may be added from user scripts.

Use the ISession.precise property to change time measurement mode.

Adding Visualizers

After device(s) have been added to the session, one or more Data Visualizers must be added. User scripts call ISession.addVisualizer or its overloads to add data visualizers to the session and optionally configure their parameters. All supported data visualizers may be added from user scripts.

Running Monitoring Session

After the session is configured, it may be started with a call to ISession.start method. Session may be paused (ISession.pause) and resumed (ISession.resume) later. To stop a monitoring session, user script

needs to call ISession.stop method. Session object becomes invalidated after this method returns and should not be used anymore.

Use the ISession.saveToLog method to execute the Save to Log command for the current session.

ISession Interface

TypeScript

```
interface ISession {
    // Properties
    readonly state: Session.State;
    precise: boolean;
    readonly visualizers: IVisualizer[];

    // Methods
    addDevice(deviceName?: string, serverName?: string): void;
    addDevice(device?: { serial: string }, serverName?: string): void;
    addDevice(device?: { usb: { device?: string; port?: number; address?: number; serialNumber?:
string; } }, serverName?: string): void;
    addDevice(device?: { network: string }, serverName?: string): void;
    addDevice(device?: { virtual: any }, serverName?: string): ISession;
    addDevice(device?: { playback: { path: string; stream?: number; } }, serverName?: string):
void;
    addDevice(device?: { bridge: string | IBridge }, serverName?: string): void;
    addVisualizer(name: string): IVisualizer;
    addVisualizer(name: "Data Recording",
        config?: VisConfig.Recorder): DataRecording.IRecordingVisualizer2;
    addVisualizer(name: "Structure View",
        structure_view_config?: { filter?: VisConfig.Filter; bind?: string; }): IVisualizer;
    addVisualizer(name: "Raw Data View",
        raw_data_view_config?: { filter?: VisConfig.Filter; bind?: string; }): IVisualizer;
    addVisualizer(name: "PPP View",
        ppp_view_config?: { filter?: VisConfig.Filter; bind?: string; }): IVisualizer;
    addVisualizer(name: "Raw Exporter",
        raw_exporter_config?: { exporter: VisConfig.Exporter; filter?: VisConfig.Filter; bind?:
string; }): IVisualizer;
    addVisualizer(name: "Text Exporter",
        text_exporter_config?: { exporter: VisConfig.Exporter; filter?: VisConfig.Filter; bind?:
string; }): IVisualizer;
    start(): void;
    stop(): void;
    pause(): void;
    resume(): void;
    setCaptureFilter(config: VisConfig.Filter): void;
    configureSource(config: { playback: Playback.Config }): void;
    configureSource(config: { path: string }): void;
    configureSource(config: { serial: Serial.CommunicationsMode; }): void;
    configureSource(config: { serial: { mode: Serial.CommunicationsMode; script?: string;
discardEmptyReads?: boolean; terminal?: IDeviceConfig; } }): void;
    configureSource(config: { multi: Multi.Color[]; }): void;
    saveToLog(config: VisConfig.Recorder): Promise<void>;
}
```

C#

```
// This interface is not available in managed environment
```

C++

```
// This interface is not available in native environment
```

ISession Properties

---

state

```typescript
readonly state: Session.State;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Returns current session state.

---

precise

```typescript
precise: boolean;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Returns or sets current session's precise timing mode. Timing mode may only be changed before the session is started.

---

visualizers

```typescript
readonly visualizers: IVisualizer[];
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Returns an array of references to data visualizer objects.

> **WARNING**
>
> Note that each time you query this property, a new object is created for each data visualizer. This means that expression `session.visualizers[0] === session.visualizers[0]` will never be true. Use the IVisualizer.equals method to test whether two references describe the same object.

ISession Methods

addDevice

```TypeScript
addDevice(deviceName?: string, serverName?: string): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`deviceName`

An optional string which is parsed by each installed data source in turn until the matched device is found. If omitted, device may later added using ISession.addDevice method.

`serverName`

Optional remote server name. If omitted, or passed empty string or ".", local monitoring session is created.

Description

Adds a specified device to the session. A monitoring session may have several devices but they all must be of the same type.

This overload uses generic string matching. It cycles through all installed modules and performs exact device name match. Serial module also supports device names of the form “COMx” where x is a number from 1 to 255.

addDevice

```TypeScript
addDevice(device?: { serial: string }, serverName?: string): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`device`

A serial device identifier (full name or COM port).

`serverName`

Optional remote server name. If omitted, or passed empty string or ".", local monitoring session is created.

Description

Adds a specified device to the session. A monitoring session may have several devices but they all must be of the same type.

---

addDevice

```TypeScript
addDevice(device?: { usb: { device?: string; port?: number; address?: number; serialNumber?: string; } }, serverName?: string): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`device`

An USB device to connect to. Empty configuration object means “Next connected device”. A combination of `device`, `port` and `address` fields use to identify the device you want to monitor. Alternatively, use the `serialNumber` field to identify the device.

`serverName`

Optional remote server name. If omitted, or passed empty string or ".", local monitoring session is created.

Description

Adds a specified device to the session. A monitoring session may have several devices but they all must be of the same type.

---

addDevice

```TypeScript
addDevice(device?: { network: string }, serverName?: string): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`device`

A network device identifier (full adapter name).

`serverName`

Optional remote server name. If omitted, or passed empty string or ".", local monitoring session is created.

Description

Adds a specified device to the session. A monitoring session may have several devices but they all must be of the same type.

---

addDevice

```
TypeScript
addDevice(device?: { virtual: any }, serverName?: string): ISession;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`device`

An object of the following type: to reference Import Data Source.

`serverName`

Optional remote server name. If omitted, or passed empty string or ".", local monitoring session is created.

Description

Adds a specified device to the session. A monitoring session may have several devices but they all must be of the same type.

---

addDevice

```
TypeScript
addDevice(device?: { playback: { path: string; stream?: number; } }, serverName?: string):
void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`device`

An object which references a given log file and optionally a stream in it.

`serverName`

Optional remote server name. If omitted, or passed empty string or ".", local monitoring session is created.

Description

Adds a specified device to the session. A monitoring session may have several devices but they all must be of the same type.

---

addDevice

```TypeScript
addDevice(device?: { bridge: string | IBridge }, serverName?: string): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`device`

A serial bridge name or a reference to IBridge interface.

`serverName`

Optional remote server name. If omitted, or passed empty string or ".", local monitoring session is created.

Description

Adds a specified device to the session. A monitoring session may have several devices but they all must be of the same type.

Example

Adding devices to the session:

```JavaScript
var session = monitoring.createSession();

// Add COM1 serial device to it (use automatic matching):
session.addDevice("COM1");

// Add specific serial device:
session.addDevice({ serial: "Communications Port (COM1)" });

// Add specific USB device:
session.addDevice({ usb: { device: "USB Input Device", port: 4, address: 1 } });

// Add specific network adapter:
session.addDevice({ network: "Realtek PCIe GBE Family Controller(802.3)" });

// Add specific log file:
session.addDevice({ playback: { path: "C:\\temp\\Communications Port (COM1)$140408$1.dmslog8",
stream: 0 } });

// Create remote monitoring session for serial port
session.addDevice("COM5", "remote_server_name");
```

addVisualizer

```TypeScript
addVisualizer(name: string): IVisualizer;
```

```C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`name`

Data visualizer name.

Description

Adds a data visualizer to the session. The method takes a name of data visualizer and an optional configuration object. Several data visualizers support configuration while some require it.

This method is overloaded to help check syntax.

---

addVisualizer

```
TypeScript
addVisualizer(name: "Data Recording",
    config?: VisConfig.Recorder): DataRecording.IRecordingVisualizer2;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`name`

Name of the Data Recording processing module.

`config`

Optional logging configuration object.

Description

Adds a Data Recording component to the session.

---

addVisualizer

```
TypeScript
addVisualizer(name: "Structure View",
    structure_view_config?: { filter?: VisConfig.Filter; bind?: string; }): IVisualizer;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`name`

Name of the Structure View data visualizer.

`structure_view_config`

Optional configuration object which configures visualizer's Display Filter (`filter` object) and Root Protocol (`bind` string). See VisConfig.Filter for more information on configuring Display Filter. Root Protocol may be configured using the `bind` parameter.

Description

Adds a Structure View data visualizer to the session.

---

addVisualizer

```TypeScript
addVisualizer(name: "Raw Data View",
    raw_data_view_config?: { filter?: VisConfig.Filter; bind?: string; }): IVisualizer;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`name`

Name of the Raw Data View data visualizer.

`raw_data_view_config`

Optional configuration object which configures visualizer's Display Filter (`filter` object) and Root Protocol (`bind` string). See VisConfig.Filter for more information on configuring Display Filter. Root Protocol may be configured using the `bind` parameter.

Description

Adds a Raw Data View data visualizer to the session.

---

addVisualizer

```TypeScript
addVisualizer(name: "PPP View",
    ppp_view_config?: { filter?: VisConfig.Filter; bind?: string; }): IVisualizer;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`name`

Name of the PPP View data visualizer.

`ppp_view_config`

Optional configuration object which configures visualizer's Display Filter (`filter` object) and Root Protocol (`bind` string). See VisConfig.Filter for more information on configuring Display Filter. Root Protocol may be configured using the `bind` parameter.

Description

Adds a PPP View data visualizer to the session.

---

addVisualizer

```TypeScript
addVisualizer(name: "Raw Exporter",
    raw_exporter_config?: { exporter: VisConfig.Exporter; filter?: VisConfig.Filter; bind?:
string; }): IVisualizer;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`name`

Name of the Raw Exporter data processing module.

`raw_exporter_config`

Configuration object must configure the exporter settings and may optionally configure Display Filter (`filter` object) and Root Protocol (`bind` string). To configure exporter, see the VisConfig.Exporter topic for more information.

Description

Adds a Raw Exporter data processing module to the session.

---

addVisualizer

```TypeScript
addVisualizer(name: "Text Exporter",
    text_exporter_config?: { exporter: VisConfig.Exporter; filter?: VisConfig.Filter; bind?:
string; }): IVisualizer;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`name`

Name of the Text Exporter data processing module.

`text_exporter_config`

Configuration object must configure the exporter settings and may optionally configure Display Filter (`filter` object) and Root Protocol (`bind` string). To configure exporter, see the VisConfig.Exporter topic for more information.

Description

Adds a Text Exporter data processing module to the session.

Example

Adding devices to the session:

```JavaScript
var session = monitoring.createSession();

// Add Request View visualizer
session.addVisualizer("Request View");

// Add Structure View visualizer with default configuration
session.addVisualizer("Structure View");

// Add PPP View visualizer and configure both Display Filter and Root Protocol
session.addVisualizer("PPP View",{filter: {name:"IO packets only"}, bind: "PPP"});

// Add Raw Exporter and configure it
session.addVisualizer("Raw Exporter",{exporter: {path: "c:\\temp\\raw.bin", overwrite: true,
nocache: false}});
```

---

start

```TypeScript
start(): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Description

Start monitoring session. Make sure at least one device and one data visualizer are successfully added to the session before calling this method. You may add devices using ISession.addDevice method and visualizers using ISession.addVisualizer.

Several data sources may support or require configuration using ISession.configureSource method.

An optional Capture Filter may be specified for monitoring session using ISession.setCaptureFilter method.

Example

Creating, configuring and starting a monitoring session:

```JavaScript
var session = monitoring.createSession();
session.addDevice({ serial: "Communications Port (COM1)" });
session.addVisualizer("Data View");
session.start();
```

stop

```typescript
TypeScript
stop(): void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Description

Stops a monitoring session. A monitoring session object may not be used after the session has been stopped. Only requesting the session state is allowed it.

pause

```typescript
TypeScript
pause(): void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Description

Pauses a running monitoring session.

resume

```typescript
TypeScript
resume(): void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Description

Resumes a paused monitoring session.

setCaptureFilter

```
TypeScript
setCaptureFilter(config: VisConfig.Filter): void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`config`

Capture Filter configuration parameter.

Description

Configure session's Capture Filter.

Example

Configuring Capture Filter:

```
JavaScript
// Use predefined capture filter
session.setCaptureFilter({ name: "IO packets only"});

// Use custom capture filter
session.setCaptureFilter({ expression: "usb.urb.BulkOrInterrupt.TransferBufferLength > 500"
});
```

---

configureSource

```
TypeScript
configureSource(config: { playback: Playback.Config }): void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`config`

Playback configuration object. See Playback.Config for more information.

Description

Configure the Playback data source.

Example

Configure Playback Source:

```JavaScript
var session = monitoring.createSession();
session.addDevice({ playback: { path: "C:\\temp\\Communications Port (COM1)$140408$1.dmslog8",
stream: 0 } });
session.configureSource({ playback: { scale: Playback.Scale.FourToOne }});
```

configureSource

```TypeScript
configureSource(config: { path: string }): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`config`

Full path to the import log file. This configures the Virtual Network Adapter source.

Description

Configure the Import Source.

configureSource

```TypeScript
configureSource(config: { serial: Serial.CommunicationsMode; }): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`config`

Serial data source communications mode. See Serial.CommunicationsMode for more information.

Description

Configure the Serial data source.

Example

```JavaScript
var session = monitoring.createSession();
session.addDevice({ serial: "Communications Port (COM1)" });
session.configureSource({ serial: Serial.CommunicationsMode.PPP});
```

configureSource

```TypeScript
configureSource(config: { serial: { mode: Serial.CommunicationsMode; script?: string;
discardEmptyReads?: boolean; terminal?: IDeviceConfig; } }): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`config`

Configuration object used to configure custom communication mode or general mode with specific value for `discardEmptyReads` (defaults to `true`). You can also configure the terminal session by passing the IDeviceConfig interface.

Description

Configure the Serial data source.

---

configureSource

```TypeScript
configureSource(config: { multi: Multi.Color[]; }): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`config`

A configuration object with color array to configure Multi-Source data source. May be combined with other configuration objects if applicable. See Multi.Color for more information.

Description

Configure the Multi-Source data source.

Example

Configure both Serial and Multi-Source:

```JavaScript
var session = monitoring.createSession();
session.addDevice({ serial: "Communications Port (COM1)" });
session.addDevice({ serial: "Next connected device" });
session.configureSource({ multi: [{ r: 250, g: 187, b: 0, a: 112 }, { r: 64, g: 64, b: 64, a:
112 }],
        serial: Serial.CommunicationsMode.PPP});
```

---

saveToLog

```TypeScript
saveToLog(config: VisConfig.Recorder): Promise<void>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`config`

Logging configuration object. At least the `path` parameter must be specified and must be the full path to the resulting log file. Other configuration parameters may be omitted, in which case defaults will be used.

Description

Start the Save to Log command on the current session object. This method pauses the monitoring session, asynchronously executes the save to log command and then resumes the session. It can throw a number of exceptions if given parameters are not valid or the session is currently in invalid state.

Example

```TypeScript
async function saveSession(session: ISession, path: string) {
        await session.saveToLog({ path: path });
}
```

IVisualizer Interface

```TypeScript
interface IVisualizer {
    // Properties
    readonly name: string;

    // Methods
    equals(object: IVisualizer): boolean;
    remove(): void;
}
```

```C#
// This interface is not available in managed environment
```

```C++
// This interface is not available in native environment
```

IVisualizer Properties

name

```TypeScript
readonly name: string;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Holds the data visualizer name.

IVisualizer Methods

equals

```TypeScript
equals(object: IVisualizer): boolean;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`object`

Reference to a data visualizer object.

Description

Checks whether two references to data visualizer objects point to the same data visualizer. Since ISession.visualizers property always creates new objects each time the user queries it, it is required to use this method to test if two references actually reference the same object.

Example

```JavaScript
var session = ...;

var vis1 = session.visualizers[0];
var vis2 = session.visualizers[0];

alert(vis1 === vis2);  // displays false
alert(vis1.equals(vis2)); // displays true
```

remove

```TypeScript
remove(): void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Description

Removes this data visualizer from the session.

Serial Namespace

Serial.CommunicationsMode Enumeration

Description

See Serial communications mode topic for more information.

Symbol	Value	Description
`Generic`	`0`	Generic (default) communications mode.
`PPP`	`1`	PPP communications mode.
`x0D`	`2`	ASCII communications mode (each packet ends with `0D` character).
`ModbusRtu`	`3`	MODBUS RTU communications mode.
`ModbusAscii`	`4`	MODBUS ASCII communications mode.
`Custom`	`5`	Custom communication mode. A TypeScript custom script with implementation of custom packet splitter must be provided.

Playback Namespace

Playback.Config Interface

Description

This interface is used to configure Playback data source.

Declaration

```
TypeScript
interface Config {
    // Properties
    range?: Playback.Range;
    scale?: Playback.Scale;
}
```

```
C#
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

Config Properties

range

```TypeScript
range?: Playback.Range;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Optional playback range configuration.

scale

```TypeScript
scale?: Playback.Scale;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Optional playback speed configuration value.

Playback.Range Interface

```TypeScript
interface Range {
    // Properties
    from: Date;
    to: Date;
}
```

```C#
public interface Range
{
    // Properties
}
```

```C++
struct Range : IDispatch
{
    // Properties
};
```

Range Properties

from

```TypeScript
from: Date;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

JavaScript `Date` object that specifies the playback starting point.

to

```TypeScript
to: Date;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

JavaScript `Date` object that specifies the playback ending point.

Playback.Scale Enumeration

Description

Playback speed.

Symbol	Value	Description
`Stepped`	`0`	Pause monitoring after each packet.
`OneToSixteen`	`1`	$^1/_{16}$ of original speed.
`OneToEight`	`2`	$^1/_8$ of original speed.
`OneToFour`	`3`	$^1/_4$ of original speed.
`OneToTwo`	`4`	$^1/_2$ of original speed.
`OneToOne`	`5`	Original speed.
`TwoToOne`	`6`	Double the original speed.
`FourToOne`	`7`	Four times the original speed.
`EightToOne`	`8`	Eight times the original speed.
`SixteenToOne`	`9`	Sixteen times the original speed.
`Continuous`	`10`	Non-stop playback. Do not make pauses between packets.

Session Namespace

Session.State Enumeration

Description

Monitoring session state.

Symbol	Value	Description
`Stopped`	`0`	Monitoring session has not been started yet.
`Running`	`1`	Monitoring session is running.
`Paused`	`2`	Monitoring session has been paused.
`Destroyed`	`3`	Monitoring session has been stopped and destroyed. Do not use this object.

Multi Namespace

Multi.Color Interface

TypeScript
```typescript
interface Color {
    // Properties
    r: number;
    g: number;
    b: number;
    a?: number;
}
```

C#
```
// This interface is not available in managed environment
```

C++
```
// This interface is not available in native environment
```

Color Properties

r

```TypeScript
r: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Red color value.

---

g

```TypeScript
g: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Green color value.

---

b

```TypeScript
b: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Blue color value.

---

a

```TypeScript
a?: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Optional alpha (transparency) color value.

VisConfig Namespace

VisConfig.Exporter Interface

Description

This interface is used to configure Raw Exporter and Text Exporter settings.

Declaration

```
TypeScript
interface Exporter {
    // Properties
    path: string;
    overwrite?: boolean;
    nocache?: boolean;
}
```

```
C#
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

Exporter Properties

---

path

```
TypeScript
path: string;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Full path to the destination file.

---

overwrite

```typescript
overwrite?: boolean;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Optional `boolean` parameter that tells if the file must be overwritten. If omitted, defaults to `false`.

---

nocache

```typescript
nocache?: boolean;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Optional `boolean` parameter that tells if disk cache should not be used. If omitted, defaults to `false`.

VisConfig.Filter Interface

Description

This interface is used to configure Display Filter or Capture Filter.

Declaration

```typescript
interface Filter {
    // Properties
    name?: string;
    expression?: string;
}
```

```csharp
// This interface is not available in managed environment
```

```cpp
// This interface is not available in native environment
```

Filter Properties

---

name

```TypeScript
name?: string;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Predefined filter name. If this property is specified, `expression` property must not be set.

---

expression

```TypeScript
expression?: string;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Custom filter expression. If this property is specified, name property must not be set.

VisConfig.Recorder Interface

```TypeScript
interface VisConfig.Recorder {
    // Properties
    path?: string;
    maxSize?: number;
    maxTimeInSeconds?: number;
    maxFiles?: number;
    overwrite?: boolean;
}
```

```C#
// This interface is not available in managed environment
```

```C++
// This interface is not available in native environment
```

VisConfig.Recorder Properties

---

path

```TypeScript
path?: string;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Full path to a log file. If omitted, the default will be used.

---

maxSize

```TypeScript
maxSize?: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Maximum size of a single log part file, in bytes. If not specified or equals zero, no size limit is enforced.

Cannot be used if `maxTimeInSeconds` is non-zero.

---

maxTimeInSeconds

```TypeScript
maxTimeInSeconds?: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Maximum length of a single log part file, in seconds. If not specified or equals zero, no time limit is enforced.

Cannot be used if `maxSize` is non-zero.

---

maxFiles

```typescript
maxFiles?: number;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Maximum number of log part files. If zero or omitted, means "infinite". Ignored if both `maxSize` and `maxTimeInSeconds` are zero.

---

overwrite

```typescript
overwrite?: boolean;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Set to `true` if existing log file must be overwritten, `false` (or do not specify) otherwise. If `overwrite` equals `false` and the destination log file exists, the logging will fail.

Ignored if `path` is empty.

DataRecording Namespace

DataRecording.IRecordingVisualizer Interface

Description

This interface provides additional methods supported by Data Recording data processing module.

Declaration

```typescript
interface IRecordingVisualizer extends IVisualizer {

    // Methods
    endStream(): void;
    newStream(): void;
}
```

```csharp
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

IRecordingVisualizer Methods

---

endStream

```
TypeScript
endStream(): void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Description

Ends the current stream and pauses session data recording.

---

newStream

```
TypeScript
newStream(): void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Description

Starts new stream and resumes session data recording.

IVisualizer

DataRecording.IRecordingVisualizer2 Interface

```
TypeScript
interface IRecordingVisualizer2 extends IVisualizer {
    // Properties
    readonly totalAmount: number;
    paused: boolean;
}
```

```
C#
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

IRecordingVisualizer2 Properties

---

totalAmount

```typescript
TypeScript
readonly totalAmount: number;
```

```csharp
C#
// This property is not available in managed environment
```

```cpp
C++
// This property is not available in native environment
```

Description

Returns the total number of bytes written to the log file.

---

paused

```typescript
TypeScript
paused: boolean;
```

```csharp
C#
// This property is not available in managed environment
```

```cpp
C++
// This property is not available in native environment
```

Description

Equals to `true` if logging is paused, `false` otherwise. Modify this property to pause/resume logging.

IVisualizer

Exporters Namespace

Exporters.IExporterVisualizer Interface

Description

This interface provides additional methods supported by Raw Exporter and Text Exporter data visualizers.

Declaration

```typescript
TypeScript
interface IExporterVisualizer extends IVisualizer {

    // Methods
    pause(): void;
    resume(): void;
}
```

```C#
// This interface is not available in managed environment
```

```C++
// This interface is not available in native environment
```

IExporterVisualizer Methods

---

pause

```TypeScript
pause(): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Description

Pauses exporting.

---

resume

```TypeScript
resume(): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Description

Resumes exporting.

IVisualizer

Serial Terminal Objects

Serial Terminal Object

The Serial Terminal object represents the Serial Terminal module to the running scripts. It exposes the ITerminalManager interface and may be access through the global object `terminal`.

User script is able to enumerate the serial terminal sessions created by the user in the Device Monitoring Studio's user interface using the ITerminalManager.sessions property. It may also create new terminal session by calling ITerminalManager.createSession method.

Device Configuration Object

This object, implementing the IDeviceConfig interface is used to configure serial terminal session parameters.

Predefined Flow Control Object

Predefined Flow Control Object has three properties: IPredefinedFlowControl.none, IPredefinedFlowControl.software and IPredefinedFlowControl.hardware that return corresponding flow control settings objects.

This object is accessible via global object `flowControl`.

Reference

ITerminalManager Interface

Description

This interface consists of methods and properties supported by the Serial Terminal module.

This interface is exposed by the global object `terminal`.

Declaration

```
TypeScript
interface ITerminalManager {
    // Properties
    readonly sessions: ITerminalSession[];

    // Methods
    closeAllSessions(): void;
    createSession(portName: string, deviceConfig?: IDeviceConfig, showWindow?: boolean):
ITerminalSession;
}
```

```
C#
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

ITerminalManager Properties

---

sessions

```
TypeScript
readonly sessions: ITerminalSession[];
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Returns the array of session objects.

Example

Obtain a list of terminal sessions and print their baud rates:

```JavaScript
var sessions = terminal.sessions;
for (var i = 0; i < sessions.length; ++i)
    alert(sessions[i].config.baudRate);
```

ITerminalManager Methods

---

closeAllSessions

```TypeScript
closeAllSessions(): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Description

Call this method to stop and close all serial terminal sessions.

---

createSession

```TypeScript
createSession(portName: string, deviceConfig?: IDeviceConfig, showWindow?: boolean):
ITerminalSession;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`portName`

The name of the device to create a session on.

`deviceConfig`

Optional device configuration object. If omitted, defaults provided by operating system are used.

`showWindow`

Pass `true` to create a visible terminal session, omit or pass `false` to create invisible session.

Return Value

Returns the reference to created empty session object.

Description

Creates new terminal session.

Example

Create new terminal session:

```JavaScript
var session = terminal.createSession("COM1", { baudRate: 115200, dataBits: 8, stopBits: 1,
        parity: Terminal.Parity.None, flowControl: flowControl.hardware }, false);
```

IDeviceConfig

IPredefinedFlowControl Interface

Description

This interface consists of properties supported by the Predefined Flow Control Object.

This interface is exposed by the global object `flowControl`.

Declaration

```TypeScript
interface IPredefinedFlowControl {
    // Properties
    none: IFlowControl;
    software: IFlowControl;
    hardware: IFlowControl;
}
```

```C#
// This interface is not available in managed environment
```

```C++
// This interface is not available in native environment
```

IPredefinedFlowControl Properties

---

none

```TypeScript
none: IFlowControl;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Returns flow control objects corresponding to “No flow control” setting.

---

software

```typescript
TypeScript
software: IFlowControl;
```

```csharp
C#
// This property is not available in managed environment
```

```cpp
C++
// This property is not available in native environment
```

Description

Returns flow control objects corresponding to “Software flow control” setting.

---

hardware

```typescript
TypeScript
hardware: IFlowControl;
```

```csharp
C#
// This property is not available in managed environment
```

```cpp
C++
// This property is not available in native environment
```

Description

Returns flow control objects corresponding to “Hardware flow control” setting.

IFlowControl Interface

Description

This interface consists of methods and properties supported by the flow control object.

Declaration

```typescript
TypeScript
interface IFlowControl {
    // Properties
    outXonXoff: boolean;
    inXonXoff: boolean;
    outCtsFlow: boolean;
    outDsrFlow: boolean;
    dsrSensitivity: boolean;
    dtrControl: Terminal.DTRControl;
    rtsControl: Terminal.RTSControl;
}
```

```
C#
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

IFlowControl Properties

---

outXonXoff

```
TypeScript
outXonXoff: boolean;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Indicates whether XON/XOFF flow control is used during transmission. If this member is true, transmission stops when the XoffChar character is received and starts again when the XonChar character is received.

---

inXonXoff

```
TypeScript
inXonXoff: boolean;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Indicates whether XON/XOFF flow control is used during reception. If this member is true, the XoffChar character is sent when the input buffer comes within XoffLim bytes of being full, and the XonChar character is sent when the input buffer comes within XonLim bytes of being empty.

---

outCtsFlow

```
TypeScript
outCtsFlow: boolean;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

If this member is true, the CTS (clear-to-send) signal is monitored for output flow control. If this member is true and CTS is turned off, output is suspended until CTS is sent again.

---

outDsrFlow

```
TypeScript
outDsrFlow: boolean;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

If this member is true, the DSR (data-set-ready) signal is monitored for output flow control. If this member is true and DSR is turned off, output is suspended until DSR is sent again.

---

dsrSensitivity

```
TypeScript
dsrSensitivity: boolean;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

If this member is true, the communications driver is sensitive to the state of the DSR signal. The driver ignores any bytes received, unless the DSR modem input line is high.

---

dtrControl

```
TypeScript
dtrControl: Terminal.DTRControl;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

The DTR (data-terminal-ready) flow control.

---

rtsControl

```
TypeScript
rtsControl: Terminal.RTSControl;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

The RTS (request-to-send) flow control.

IDeviceConfig Interface

Description

This interface contains properties used to configure terminal session object.

Declaration

```
TypeScript
interface IDeviceConfig {
    // Properties
    baudRate?: number;
    dataBits?: Terminal.DataBits;
    stopBits?: Terminal.StopBits;
    parity?: Terminal.Parity;
    flowControl?: IFlowControl;
    timeouts?: ISerialTimeouts;
}
```

```
C#
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

IDeviceConfig Properties

---

baudRate

```
TypeScript
baudRate?: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Terminal session baud rate.

---

dataBits

```
TypeScript
dataBits?: Terminal.DataBits;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

The number of bits per byte.

---

stopBits

```
TypeScript
stopBits?: Terminal.StopBits;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

The number of stop bits.

---

parity

```
TypeScript
parity?: Terminal.Parity;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Parity mode.

---

flowControl

```
TypeScript
flowControl?: IFlowControl;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Flow control mode.

---

timeouts

```
TypeScript
timeouts?: ISerialTimeouts;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Session timeout values.

IFlowControl

ISerialTimeouts Interface

Description

This interface is used to set session's timeout values.

A read or write request successfully completes when either the specified number of bytes is transferred or the requested read or write operation times out. The request returns the `STATUS_SUCCESS` status code to indicate that the specified number of bytes was transferred. The request returns the `STATUS_TIMEOUT` status code to indicate that the operation timed out.

For a read operation that is N_{total} bytes in length, the maximum amount of time, T_{max}, that the serial port allows for the operation to complete is calculated as follows:

$T_{max} = N_{total} *$ `readTotalTimeoutMultiplier` + `readTotalTimeoutConstant`

A read request that exceeds this maximum completes when the time-out occurs, and throws a timeout

exception.

For a write operation that is N_{total} bytes in length, the maximum amount of time, T_{max}, that the serial port allows for the operation to complete is calculated as follows:

$T_{max} = N_{total} *$ `writeTotalTimeoutMultiplier` + `writeTotalTimeoutConstant`

A write request that exceeds this maximum completes when the time-out occurs, and throws a timeout exception.

The maximum time, T_{max}, that is allowed for a read or write operation to complete is always measured from when the serial port starts the requested operation, and not from when the client submits the request.

If readIntervalTimeout, `readTotalTimeoutMultiplier`, and `readTotalTimeoutConstant` are all zero, read operations never time out. If writeTotalTimeoutMultiplier and writeTotalTimeoutConstant are both zero, write operations never time out.

If readIntervalTimeout is zero, there is no maximum interval between consecutive bytes in read operations, and time-outs are based solely on the `readTotalTimeoutMultiplier` and `readTotalTimeoutConstant` members.

If both `readTotalTimeoutMultiplier` and `readTotalTimeoutConstant` are zero, and `readIntervalTimeout` is less than `0xFFFFFFFF` and greater than zero, a read operation times out only if the interval between a pair of consecutively received bytes exceeds readIntervalTimeout. If these three time-out values are used, and the serial port's input buffer is empty when a read request is sent to the port, this request never times out until after the port receives at least one byte of new data.

If readIntervalTimeout is set to `0xFFFFFFFF`, and both `readTotalTimeoutConstant` and `readTotalTimeoutMultiplier` are zero, a read request completes immediately with the bytes that have already been received, even if no bytes have been received.

If both readIntervalTimeout and `readTotalTimeoutMultiplier` are set to `0xFFFFFFFF`, and `readTotalTimeoutConstant` is set to a value greater than zero and less than `0xFFFFFFFF`, a read request behaves as follows:

- If there are any bytes in the serial port's input buffer, the read request completes immediately with the bytes that are in the buffer.
- If there are no bytes in the input buffer, the serial port waits until a byte arrives, and then immediately completes the read request with the one byte of data.
- If no bytes arrive within the time specified by `readTotalTimeoutConstant`, the read request times out throwing a timeout exception.

Declaration

TypeScript

```typescript
interface ISerialTimeouts {
    // Properties
    readIntervalTimeout?: number;
    `readTotalTimeoutMultiplier`?: number;
    `readTotalTimeoutConstant`?: number;
    writeTotalTimeoutMultiplier?: number;
    writeTotalTimeoutConstant?: number;
}
```

C#

```
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

ISerialTimeouts Properties

---

readIntervalTimeout

```
TypeScript
readIntervalTimeout?: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

The maximum amount of time, in milliseconds, that is allowed between two consecutive bytes in a read operation. A read operation that exceeds this maximum times out. This maximum does not apply to the time interval that precedes the reading of the first byte. A value of zero indicates that interval time-outs are not used.

`readTotalTimeoutMultiplier`

```
TypeScript
`readTotalTimeoutMultiplier`?: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

The maximum amount of time, in milliseconds, that is allowed per byte in a read operation. A read operation that exceeds this maximum times out.

`readTotalTimeoutConstant`

```
TypeScript
`readTotalTimeoutConstant`?: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

The maximum amount of additional time, in milliseconds, that is allowed per read operation. A read operation that exceeds this maximum times out.

---

writeTotalTimeoutMultiplier

```
TypeScript
writeTotalTimeoutMultiplier?: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

The maximum total time, in milliseconds, that is allowed per byte in a write operation. A write operation that exceeds this maximum times out.

---

writeTotalTimeoutConstant

```
TypeScript
writeTotalTimeoutConstant?: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

The maximum amount of additional time, in milliseconds, that is allowed per write operation. A write operation that exceeds this maximum times out.

Serial Terminal Session Object

Serial Terminal Session object represents a running terminal session. It implements the ITerminalSession interface. A reference to a session object is obtained through the ITerminalManager.sessions property or ITerminalManager.createSession method.

Configuring Terminal Session

ITerminalManager.createSession receives a session configuration object, which is provided by IDeviceConfig interface. Session's configuration object may be accessed using the property ITerminalSession.config. Once the session is started, this property is read-only and any changes made to the session configuration object's properties are ignored.

You can create a visible terminal session (a session with designated window in UI is provided) or a hidden session (controlled by the ITerminalSession.visible property).

Starting and Stopping Terminal Session

To start a configured session, user script calls the ITerminalSession.start method. To stop, it calls the ITerminalSession.stop method. When the session is stopped, a serial device is closed and may be used by any other application.

Sending Data

Terminal Session object provides a number of methods to send data to the serial device:

Method	Description
ITerminalSession.send	Generic method that is overloaded to send an ASCII string, a single byte or a byte array.
ITerminalSession.sendAs	Send a given integer array, treating its members as 8-bit, 16-bit, 32-bit or 64-bit integers, with either little-endian or big-endian encoding.
ITerminalSession.sendFile	Send the contents of a given file.

All send methods return a promise which must be awaited before trying to send any more data in order to prevent output buffer overflow. If any error occurs, the send method throws an exception.

Receiving Data

To receive data from serial port, call the ITerminalSession.receive method. It returns a promise which produces a byte array with available data. Alternatively, a user script may subscribe to the ITerminalSession.received event.

Events

Terminal Session exposes two events: ITerminalSession.sent and ITerminalSession.received. They are fired when new data is sent or received from the serial device correspondingly.

Flow Control Emulation

A terminal session provides access to various serial device flow control states. See the corresponding properties and methods of ITerminalSession interface.

Reference

ITerminalSession Interface

Description

This interface is implemented by the terminal session object. You receive a session object either from ITerminalManager.sessions array, or through a call to ITerminalManager.createSession method.

Declaration

```typescript
interface ITerminalSession {
    // Properties
    friendlyName: string;
    portName: string;
    config: IDeviceConfig;
    rts: boolean;
    dtr: boolean;
    readonly cts: boolean;
    readonly dsr: boolean;
    readonly dcd: boolean;
    readonly ring: boolean;
    readonly visible: boolean;

    // Methods
    xon(): void;
    xoff(): void;
    breakOn(): void;
    breakOff(): void;
    start(): void;
    stop(): void;
    send(text: string): Promise<void>;
    send(byte: number): Promise<void>;
    send(bytes: number [] | Uint8Array | Uint16Array | Uint32Array | ArrayBuffer | DataView):
Promise<void>;
    sendAs(sendAs: Terminal.SendAs, data: number [] | Uint8Array | Uint16Array | Uint32Array |
ArrayBuffer | DataView, bigEndian?: boolean): Promise<void>;
    sendFile(fileName: string, byLines: boolean): Promise<void>;
    receive(): Promise<Uint8Array>;

    // Events
    sent(handler: (data: Uint8Array) => void): number;
    sent(eventId: number): void;
    received(handler: (data: Uint8Array) => void): number;
    received(eventId: number): void;
}
```

```
// This interface is not available in managed environment
```

```
// This interface is not available in native environment
```

ITerminalSession Properties

---

friendlyName

```typescript
friendlyName: string;
```

```
// This property is not available in managed environment
```

```
// This property is not available in native environment
```

Description

The session friendly name.

portName

```
TypeScript
portName: string;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

The property stores the so-called device interface string for a serial device. The device interface is one of the following:

- The string of format “COMn”, where n is the port number. It is internally converted to a string “\.\COMn”.
- The string of format “\.\COMn”, which is equivalent to the previous.
- The device interface string of format "\?\acpi#pnp0501#1#{0100fdd7-be5a-4808-91f5-05002bc60f72}", which uniquely identifies any device in the system. The various tools, including Device Monitoring Studio itself may be used to obtain device interfaces for installed devices.

config

```
TypeScript
config: IDeviceConfig;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Get or set a terminal session configuration object. See IDeviceConfig interface for details. Any configuration object property may be omitted, in which case the system default value is used.

rts

```
TypeScript
rts: boolean;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Ready To Send signal state (RTS).

dtr

```TypeScript
dtr: boolean;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Data Terminal Ready signal state (DTR).

cts

```TypeScript
readonly cts: boolean;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Clear To Send signal state (CTS). This property is read-only.

dsr

```TypeScript
readonly dsr: boolean;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Data Set Ready signal state (DSR). This property is read-only.

dcd

```typescript
readonly dcd: boolean;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Data Carrier Detect (DCD). This property is read-only.

---

ring

```typescript
readonly ring: boolean;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Ring Indicator (RI). This property is read-only.

---

visible

```typescript
readonly visible: boolean;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

`true` if this session is visible in UI, `false` otherwise. This property is read-only.

ITerminalSession Methods

---

xon

```typescript
xon(): void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Description

Simulate that XON character received.

---

xoff

```
TypeScript
xoff(): void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Description

Simulate that XOFF character received.

---

breakOn

```
TypeScript
breakOn(): void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Description

Simulate BREAK ON state.

---

breakOff

```
TypeScript
breakOff(): void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Description

Simulate BREAK OFF state.

---

start

```TypeScript
start(): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Description

Start a terminal session.

---

stop

```TypeScript
stop(): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Description

Stop a terminal session. This will cause the opened handle to a serial device to be closed.

---

send

```TypeScript
send(text: string): Promise<void>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`text`

String to be sent to the terminal session.

Description

Send data to a terminal session. A given string is converted to ANSI and sent to the port.

---

send

```TypeScript
send(byte: number): Promise<void>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`byte`

A single byte to be sent to the terminal session.

Description

Send a given byte to a terminal session.

---

send

```TypeScript
send(bytes: number [] | Uint8Array | Uint16Array | Uint32Array | ArrayBuffer | DataView):
Promise<void>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`bytes`

A JavaScript array of integer numbers to be sent to the terminal session.

Description

Send a passed byte array to a terminal session.

---

sendAs

```TypeScript
sendAs(sendAs: Terminal.SendAs, data: number [] | Uint8Array | Uint16Array | Uint32Array |
ArrayBuffer | DataView, bigEndian?: boolean): Promise<void>;
```

```C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`sendAs`

@split A constant that tells how to interpret numbers in an array. See Terminal.SendAs for more information.

`data`

A JavaScript array of integer numbers to send to the port. Numbers are interpreted according to `sendAs` parameter.

`bigEndian`

An optional `boolean` parameter, which if passed and equals to true tells terminal session to use big-endian encoding for integers. Ignored for when `sendAs` equals `Terminal.SendAs.Bytes`. Equals `false` if omitted.

Description

Send a number array to terminal session. First parameter tells how to interpret numbers in an array and the second parameter is a Javascript array of integer numbers.

Example

Send data to the terminal session

```
JavaScript
await session.sendAs(Terminal.SendAs.Words, [ 0x4154, 0x440d ], true);
```

---

sendFile

```
TypeScript
sendFile(fileName: string, byLines: boolean): Promise<void>;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`fileName`

Full path to a file to send. Specify empty string to open dialog box, that will allow you to visually select file using system Open File dialog.

`byLines`

If this parameter is `true`, the file is analyzed and split into the lines. The lines are then sent to the serial terminal session. If this parameter is `false`, the file is sent to the session byte by byte.

Description

Send contents of a file to the terminal session.

---

receive

```TypeScript
receive(): Promise<Uint8Array>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Return Value

A promise object that produces a byte array when ready.

Description

Receive (read) data from the terminal session.

Example

Receive data from the terminal session

```JavaScript
async function check(session)
{
  await session.send("AT\n");
  var response = await session.receive();
}
```

ITerminalSession Events

sent

```TypeScript
sent(handler: (data: Uint8Array) => void): number;
sent(eventId: number): void;
```

Parameters

`data`

Sent bytes in JavaScript array.

Description

This event is fired when new data is sent to the serial device.

First overload is used to bind new handler to the event. It returns a numeric `eventId` which then may be passed to second overload to unbind a handler.

Example

Sample sent data handler for a text protocol.

```
var user_session = terminal.sessions[0];
var eventId = user_session.sent(function(data)
{
  alert(data.length + " bytes sent to " + user_session.FriendlyName + ".\n");
});

// ...
// Unbind event handler when not needed anymore
user_session.sent(eventId);
```

---

received

```
TypeScript
received(handler: (data: Uint8Array) => void): number;
received(eventId: number): void;
```

Parameters

`data`

Received bytes in JavaScript array.

Description

This event is fired when new data is received from the serial device.

First overload is used to bind new handler to the event. It returns a numeric `eventId` which then may be passed to second overload to unbind a handler.

Example

Sample received data handler for a text protocol.

```
JavaScript
var user_session = terminal.sessions[0];
var eventId = user_session.received(function(data)
{
  alert(data.length + " bytes received to " + user_session.FriendlyName + ".\n");
});

// ...
// Unbind event handler when not needed anymore
user_session.received(eventId);
```

IDeviceConfig

Terminal Namespace

Terminal.DataBits Enumeration

Description

Allowed data bits constants.

Symbol	Value	Description
`_5`	`5`	5 bits data length.
`_6`	`6`	6 bits data length.
`_7`	`7`	7 bits data length.
`_8`	`8`	8 bits data length.

Terminal.Parity Enumeration

Description

Allowed parity constants.

Symbol	Value	Description
`None`	`0`	No parity.
`Odd`	`1`	Odd parity.
`Even`	`2`	Even parity.
`Mark`	`3`	Mark parity.
`Space`	`4`	Space parity.

Terminal.SendAs Enumeration

Description

Allowed array types for ITerminalSession.sendAs method.

Symbol	Value	Description
`Bytes`	`0`	Treat array members as unsigned 8-bit integers.
`Words`	`1`	Treat array members as unsigned 16-bit integers.
`DoubleWords`	`2`	Treat array members as unsigned 32-bit integers.

Terminal.StopBits Enumeration

Description

Allowed stop bits constants.

Symbol	Value	Description
`_1`	`0`	1 stop bit.
`_1_5`	`1`	1.5 stop bits.
`_2`	`2`	2 stop bits.

Terminal.DTRControl Enumeration

Description

Supported DTR (data-terminal-ready) control modes.

Symbol	Value	Description
`Disabled`	`0`	Disables the DTR line when the device is opened and leaves it disabled.
`Enabled`	`1`	Enables the DTR line when the device is opened and leaves it on.
`Handshake`	`2`	Enables DTR handshaking.

Terminal.RTSControl Enumeration

Description

Supported RTS (request-to-send) control modes.

Symbol	Value	Description
`Disabled`	`0`	Disables the RTS line when the device is opened and leaves it disabled.
`Enabled`	`1`	Enables the RTS line when the device is opened and leaves it on.
`Handshake`	`2`	Enables RTS handshaking. The driver raises the RTS line when the "type-ahead" (input) buffer is less than one-half full and lowers the RTS line when the buffer is more than three-quarters full.
`Toggle`	`3`	Specifies that the RTS line will be high if bytes are available for transmission. After all buffered bytes have been sent, the RTS line will be low.

Network Manager Object

Network Manager object is used to create TCP sessions, UDP sessions and TCP Listeners. TCP sessions may then be used with MODBUS Send module to build and send MODBUS TCP packets to destination endpoints.

Network Manager is exposed by global object `networkManager` and implements INetworkManager interface. Use this object to enumerate all running sessions (INetworkManager.getSessions), and create new sessions and listeners:

- TCP sessions (INetworkManager.createTcpSession).
- UDP sessions (INetworkManager.createUdpSession).
- TCP Listener objects (INetworkManager.createTcpListener).

Reference

INetworkManager Interface

```
TypeScript
interface INetworkManager {

    // Methods
    createTcpSession(ipAddress: string, port: number): ITcpSession;
    createUdpSession(): IUdpSession;
    createTcpListener(host: string, port: number): Promise<ITcpListener>;
    getSessions(type: Network.SessionType): INetworkSession[];
}
```

```
C#
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

INetworkManager Methods

---

createTcpSession

```
TypeScript
createTcpSession(ipAddress: string, port: number): ITcpSession;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`ipAddress`

A destination address. This may be either IPv4 address, IPv6 address or DNS name.

`port`

Destination port number.

Return Value

Returns the reference to created session object.

Description

Creates new TCP session.

Example

Create new TCP session:

```
JavaScript
var session = networkManager.createTcpSession("192.168.33.50", 502);
```

---

createUdpSession

```
TypeScript
createUdpSession(): IUdpSession;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Return Value

Returns the reference to created session object.

Description

Creates new UDP session.

Example

Create new UDP session:

```JavaScript
var session = networkManager.createUdpSession();
```

---

createTcpListener

```TypeScript
createTcpListener(host: string, port: number): Promise<ITcpListener>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`host`

A local host name. Used to open a listening socket.

`port`

Local port number.

Return Value

Returns a promise that yields a reference to the prepared listener object.

Description

Creates new TCP listener object.

Example

Listen for a new TCP connection

```JavaScript
async function listen() {
    var l = await networkManager.createTcpListener("localhost", 80);
    var socket = await l.listen();
}
```

---

getSessions

```TypeScript
getSessions(type: Network.SessionType): INetworkSession[];
```

```C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`type`

Type of sessions to return.

Return Value

Array of requested session objects.

Description

Retrieve the array of TCP or UDP (or both) session objects.

Example

Obtain a list of TCP sessions.

```
JavaScript
var tcpSessions = networkManager.getSessions(Network.SessionType.Tcp);
```

Network Namespace

Network.SessionType Enumeration

Description

A constant that controls the type of sessions the INetworkManager.getSessions method returns.

Symbol	Value	Description
`Tcp`	`1`	Return only TCP sessions.
`Udp`	`2`	Return only UDP sessions.
`All`	`3`	Return both TCP and UDP sessions.

INetworkSession Interface

```
TypeScript
interface INetworkSession {

    // Methods
    connect(hostName: string, port: number): Promise<void>;
    stop(): void;
    send(text: string): void;
    send(byte: number): void;
    send(bytes: number [] | Uint8Array | Uint16Array | Uint32Array | ArrayBuffer | DataView):
void;
    receive(): Promise<Uint8Array>;
}
```

```
C#
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

INetworkSession Methods

connect

```TypeScript
connect(hostName: string, port: number): Promise<void>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`hostName`

Host name. This may be host domain name, IP address or other name that can be resolved.

`port`

Port number.

Description

Starts a connect operation on a session. Returns a promise that can be awaited. A promise object is completed when the connection either succeeds or fails.

stop

```TypeScript
stop(): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Description

Stop the running TCP session.

send

```TypeScript
send(text: string): void;
```

```C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`text`

String to be sent to the terminal session.

Description

Send data to a TCP session. Convert a given string to ANSI and sends it to the session.

---

send

```
TypeScript
send(byte: number): void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`byte`

A single byte to be sent to the terminal session.

Description

Send a given byte to a TCP session.

---

send

```
TypeScript
send(bytes: number [] | Uint8Array | Uint16Array | Uint32Array | ArrayBuffer | DataView):
void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`bytes`

A JavaScript array of integer numbers to be sent to the terminal session. Each number is treated as an unsigned 8-bit value.

Description

Send given byte array to a TCP session.

Example

Send data to the terminal session

```JavaScript
// Send ASCII string to the TCP endpoint
session.send("AT\n");

// Send a single byte to the TCP endpoint
session.send(0x0d);

// Send a byte array to the TCP endpoint
session.send([ 0x41, 0x54, 0x0d ]);
```

---

receive

```TypeScript
receive(): Promise<Uint8Array>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Description

Starts a receive operation on a network session. This method returns a promise that yields a byte array when completed.

Example

Receive data from the network

```JavaScript
async function read(session: ITcpSession)
{
    while (true)
    {
        try
        {
            var data = await session.receive();
            // process data (Uint8Array)
        } catch(e)
        {
            // network error occurred
        }
    }
}
```

TCP Session Object

TCP Session object represents a running TCP session. It implements theITcpSession interface. A reference to a session object is obtained through the call to INetworkManager.createTcpSession method.

Asynchronous API

TCP Session API is asynchronous. All methods (exceptITcpSession.stop) return promise objects. This

means that they complete immediately and the caller needs to schedule a continuation to get the result of the operation. This can be done using the call to `Promise.then` or using ES2017 `await` keyword.

Connecting a TCP Session

Before the session can be used for data transfer, it must first be *connected* to remote endpoint by a call to ITcpSession.connect method. This method returns a promise that is completed when the connection succeeds or fails.

Sending and Receiving Data

TCP Session object implements the ITcpSession.send method with three overloads that allows the user script to send the data to the TCP session. Use ITcpSession.receive method to read data from a TCP session.

ITcpSession Interface

```
TypeScript
interface ITcpSession extends INetworkSession {
}
```

```
C#
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

INetworkSession

UDP Session Object

UDP Session object represents a running UDP session. It implements theIUdpSession interface. A reference to a session object is obtained through the INetworkManager.createUdpSession method.

Asynchronous API

UDP Session API is asynchronous. All methods (exceptIUdpSession.stop) return promise objects. This means that they complete immediately and the caller needs to schedule a continuation to get the result of the operation. This can be done using the call to `Promise.then` or using ES2017 `await` keyword.

Starting and Stopping UDP Session

Client UDP session should be started with a call toIUdpSession.connect method. It returns a promise that is completed as soon as connection is established. Server UDP session should be started with a call to IUdpSession.bind method.

To stop a session, call IUdpSession.stop method. When UDP session object is collected by garbage collector, the session is closed automatically.

Sending and Receiving Data

UDP Session object implements theIUdpSession.send method with three overloads that allows the user script to send the data to the UDP session. Method returns promise that completed as soon as data sending is finished.

To receive data from a session, call IUdpSession.receive method. It returns a promise object (`Promise<Uint8Array>`).

IUdpSession Interface

```TypeScript
interface IUdpSession extends INetworkSession {

    // Methods
    bind(hostName: string, port: number): Promise<void>;
}
```

```C#
// This interface is not available in managed environment
```

```C++
// This interface is not available in native environment
```

IUdpSession Methods

---

bind

```TypeScript
bind(hostName: string, port: number): Promise<void>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`hostName`

Host name. This may be host domain name, IP address or other name that can be resolved.

`port`

Port number.

Description

Starts a bind operation on UDP session. Returns a promise that can be awaited.

INetworkSession

TCP Listener Object

TCP Listener object represents a listening TCP socket and implements theITcpListener interface. You create an instance of a listener object by calling the INetworkManager.createTcpListener method. To start listening on a socket, call the ITcpListener.listen method. The method returns a promise that, when ready, provides the actual TCP session to use to communicate with a connected party.

ITcpListener Interface

```typescript
interface ITcpListener {

    // Methods
    listen(): Promise<ITcpSession>;
    close(): void;
}
```

```csharp
// This interface is not available in managed environment
```

```cpp
// This interface is not available in native environment
```

ITcpListener Methods

listen

```typescript
listen(): Promise<ITcpSession>;
```

```csharp
// This method is not available in managed environment
```

```cpp
// This method is not available in native environment
```

Return Value

Returns a promise. When the promise becomes ready, it produces a reference to TCP Session object you may use to communicate with connected party.

Description

Start listening on a configured endpoint. This method completes immediately, returning a promise object. When this promise becomes ready, it yields a reference to new TCP session object, which represents the connected party.

Example

Listen for a new TCP connection

```javascript
async function listen() {
    var l = await networkManager.createTcpListener("localhost", 80);
    var socket = await l.listen();
}
```

close

```typescript
close(): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Description

Closes the listener object. This object may not be used after the close operation.

ITcpSession

MODBUS Manager Object

The MODBUS Manager object represents theMODBUS Send Module to the running scripts. It is available as a global object `modbus` and implements the IModbusManager interface. The only method of this interface, IModbusManager.createBuilder is used to create a MODBUS Builder Object.

IModbusManager Interface

```TypeScript
interface IModbusManager {

    // Methods
    createBuilder(address: number, asciiMode: boolean): IModbusBuilder;
}
```

```C#
// This interface is not available in managed environment
```

```C++
// This interface is not available in native environment
```

IModbusManager Methods

---

createBuilder

```TypeScript
createBuilder(address: number, asciiMode: boolean): IModbusBuilder;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`address`

MODBUS destination address.

`asciiMode`

True if ASCII protocol is required, false for RTU.

Description

This method creates new MODBUS Builder Object It accepts an address and protocol mode. The created builder object will internally store these values and will then use them in construction of MODBUS requests and responses.

MODBUS Builder Object

MODBUS Builder object is used to construct MODBUS request and response messages. This object exposes a number of helper methods (via IModbusBuilder interface) that should be called to parse passed arguments and build message's byte array. The resulting byte array may later be used in a call to ITerminalSession.send, ITcpSession.send or IUdpSession.send methods.

Creating MODBUS Builder Object

To create a MODBUS Builder object, call the IModbusManager.createBuilder method.

Reference

IModbusBuilder Interface

Description

This interface is implemented by the MODBUS Builder Object You create an instance of this object by calling the IModbusManager.createBuilder method.

Please consult the MODBUS protocol documentation for term definitions.

Declaration

```typescript
interface IModbusBuilder {

    // Methods
    error(functionID: number, exceptionCode: number): Uint8Array;
    requestDiagnostics(subfunction: number, data: number): Uint8Array;
    requestGetCommEventCounter(): Uint8Array;
    requestGetCommEventLog(): Uint8Array;
    requestMaskWriteRegister(referenceAddress: number, andMask: number, orMask: number):
Uint8Array;
    requestReadCoils(startingAddress: number, numberOfCoils: number): Uint8Array;
    requestReadDiscreteInputs(startingAddress: number, numberOfInputs: number): Uint8Array;
    requestReadExceptionStatus(): Uint8Array;
    requestReadFIFOQueue(fifoPointerAddress: number): Uint8Array;
    requestReadFileRecord(requests: ReadFileRequest[]): Uint8Array;
    requestReadHoldingRegisters(startingAddress: number, numberOfRegisters: number): Uint8Array;
    requestReadInputRegisters(startingAddress: number, numberOfRegisters: number): Uint8Array;
    requestReadWriteMultipleRegisters(readStartingAddress: number,
        numberToRead: number,
        writeStartingAddress: number,
        data: number [] | Uint16Array): Uint8Array;
    requestReportSlaveID(): Uint8Array;
    requestUserFunction(functionID: number, data: number[] | Uint8Array): Uint8Array;
    requestWriteFileRecord(requests: FileRecord[]): Uint8Array;
    requestWriteMultipleCoils(startingAddress: number, data: boolean[]): Uint8Array;
    requestWriteMultipleRegisters(startingAddress: number, data: number[] | Uint16Array):
Uint8Array;
    requestWriteSingleCoil(outputAddress: number, outputValue: number): Uint8Array;
    requestWriteSingleRegister(registerAddress: number, registerValue: number): Uint8Array;
    responseDiagnostics(subFunction: number, data: number [] | Uint8Array): Uint8Array;
    responseGetCommEventCounter(status: number, eventCount: number): Uint8Array;
    responseGetCommEventLog(status: number, events: number [] | Uint8Array): Uint8Array;
    responseMaskWriteRegister(referenceAddress: number, andMask: number, orMask: number):
Uint8Array;
    responseReadCoilStatus(items: number [] | Uint8Array): Uint8Array;
    responseReadDiscreteInputs(items: number [] | Uint8Array): Uint8Array;
    responseReadExceptionStatus(outputData: number): Uint8Array;
    responseReadFIFOQueue(registers: number [] | Uint16Array): Uint8Array;
    responseReadFileRecord(requests: ReadFileResponse[]): Uint8Array;
    responseReadHoldingRegisters(registers: number [] | Uint16Array): Uint8Array;
    responseReadInputRegisters(registers: number [] | Uint16Array): Uint8Array;
    responseReadWriteRegisters(registers: number [] | Uint16Array): Uint8Array;
    responseReportSlaveID(runIndicatorStatus: number,
        slaveIds: number [] | Uint8Array,
        data: number [] | Uint8Array): Uint8Array;
    responseWriteFileRecord(requests: FileRecord[]): Uint8Array;
    responseWriteMultipleCoils(startingAddress: number, quantityOfOutputs: number): Uint8Array;
    responseWriteMultipleRegisters(startingAddress: number, quantityOfRegisters: number):
Uint8Array;
    responseWriteSingleCoil(outputAddress: number, outputValue: number): Uint8Array;
    responseWriteSingleRegister(outRegisterAddress: number, outValue: number): Uint8Array;
}
```

```csharp
// This interface is not available in managed environment
```

```cpp
// This interface is not available in native environment
```

IModbusBuilder Methods

---

error

```typescript
error(functionID: number, exceptionCode: number): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`functionID`

Function code (ID). The function code is a number from 0 to 127. Internally, the MODBUS Send module appends 0x80 to it.

`exceptionCode`

Exception code (see MODBUS protocol documentation).

Description

Construct the error response.

Example

```JavaScript
var message = builder.error(1, 20);
```

---

requestDiagnostics

```TypeScript
requestDiagnostics(subfunction: number, data: number): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`subfunction`

Diagnostics sub-function code (0 to 65535). Please refer to the protocol documentation for a list of predefined sub-function codes.

`data`

Data to be returned (looped back) in the device response. Returned data should match the original for the request to be considered successful.

Description

Send the Diagnostics (see MODBUS protocol documentation) request to the selected device.

Example

```JavaScript
var message = builder.requestDiagnostics(1, 65535);
```

requestGetCommEventCounter

```TypeScript
requestGetCommEventCounter(): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Description

Construct the Get Comm Event Counter request.

Example

```JavaScript
var message = builder.requestGetCommEventCounter();
```

requestGetCommEventLog

```TypeScript
requestGetCommEventLog(): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Description

Construct the Get Comm Event Log request.

requestMaskWriteRegister

```TypeScript
requestMaskWriteRegister(referenceAddress: number, andMask: number, orMask: number):
Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`referenceAddress`

Reference address. It can be a number from 0 to 65535.

`andMask`

And Mask is used for bitwise AND operation on the selected register address. It can be a number from 0 to 65535.

`orMask`

And Mask is used for bitwise OR operation on the selected register address. It can be a number from 0 to 65535.

Description

Construct the Mask Write Register request.

---

requestReadCoils

```
TypeScript
requestReadCoils(startingAddress: number, numberOfCoils: number): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`startingAddress`

First coil address.

`numberOfCoils`

Total number of coils to read. This parameter can take values from 1 to 65535 (0x0001 to 0xFFFF). However, the documentation allows only 1 to 2000 (0x0001 to 0x07D0) values to be used (see the MODBUS protocol documentation).

Description

Construct the Read Coils request.

---

requestReadDiscreteInputs

```
TypeScript
requestReadDiscreteInputs(startingAddress: number, numberOfInputs: number): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`startingAddress`

First discrete input address. In some implementations device interprets this value as an offset - this value is added to default MODBUS base discrete input address (see MODBUS protocol

documentation) 10000. Specify the 0 value for this parameter to read discrete input with 10000 address.

`numberOfInputs`

Total count of discrete inputs to be read. This parameter can take values from 1 to 65535 (0x0001 to 0xFFFF). However, the documentation allows only 1 to 2000 (0x0001 to 0x07D0) values to be used (see the MODBUS protocol documentation).

Description

Construct the Read Discrete Inputs request.

---

requestReadExceptionStatus

```
TypeScript
requestReadExceptionStatus(): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Description

Construct the Read Exception Status request.

---

requestReadFIFOQueue

```
TypeScript
requestReadFIFOQueue(fifoPointerAddress: number): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`fifoPointerAddress`

Queue pointer address. This parameter can take values from 0 to 65535 (0x0000 to 0xFFFF).

Description

Construct the Read FIFO Queue request.

---

requestReadFileRecord

```
TypeScript
requestReadFileRecord(requests: ReadFileRequest[]): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`requests`

Array of read file requests.

Description

Construct the Read File Record request.

Example

```
JavaScript
var message = builder.requestReadFileRecord([
  { referenceType: 1, fileNumber: 10, recordNumber: 20, registerLength: 30 }
  ]); // 1 request with refernce type 1 and 10 as file number value

var message = builder.requestReadFileRecord([
  { referenceType: 1, fileNumber: 10, recordNumber: 20, registerLength: 30 },
  { referenceType: 1, fileNumber: 11, recordNumber: 21, registerLength: 31 }
  ]); // 2 requests: first with 20 as the Record Number value; second with 21 as the Record
Number value
```

---

requestReadHoldingRegisters

```
TypeScript
requestReadHoldingRegisters(startingAddress: number, numberOfRegisters: number): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`startingAddress`

First register address (0 to 65535). In some implementations device interprets this value as an offset - this value is added to default MODBUS base register address (see Modbus protocol documentation) 40000. Specify the 0 value for this parameter to write registers starting from 40000 address.

`numberOfRegisters`

Total count of registers to read. This parameter can take values from 1 to 65535 (0x0001 to 0xFFFF). However, the documentation allows only 1 to 125 (0x0001 to 0x007D) values to be used (see the MODBUS protocol documentation).

Description

Construct the Read Holding Registers request.

---

requestReadInputRegisters

```
TypeScript
requestReadInputRegisters(startingAddress: number, numberOfRegisters: number): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`startingAddress`

First register address (0 to 65535). In some implementations device interprets this value as an offset - this value is added to default MODBUS base register address (see MODBUS protocol documentation) 30000. Specify the 0 value for this parameter to write registers starting from 30000 address.

`numberOfRegisters`

Total number of registers to read. This parameter can take values from 1 to 65535 (0x0001 to 0xFFFF). However, the documentation allows only 1 to 125 (0x0001 to 0x007D) values to be used (see the MODBUS protocol documentation).

Description

Construct the Read Input Registers request.

---

requestReadWriteMultipleRegisters

```
TypeScript
requestReadWriteMultipleRegisters(readStartingAddress: number,
    numberToRead: number,
    writeStartingAddress: number,
    data: number [] | Uint16Array): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`readStartingAddress`

Read starting address (8-bit).

`numberToRead`

Number of values to read (8-bit).

`writeStartingAddress`

Write starting address (8-bit).

`data`

Data to write. Treated as unsigned 16-bit integers.

Description

Construct the Read Write Multiple Registers request.

The function throws an exception if you specify incorrect values for the `data` parameter. The function also throws an exception if there is too many or too little values, as well as if the total length of the packet exceeds 255 bytes.

Example

```
JavaScript
var message1 = builder.requestReadWriteMultipleRegisters(1, 2, 30, [ 32500 ]);  // only one
value is written
var message2 = builder.requestReadWriteMultipleRegisters(1, 2, 30, [ 32500, 32000, 65535,
65535, 65535 ]);  // 5 values are written
```

---

requestReportSlaveID

```
TypeScript
requestReportSlaveID(): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Description

Construct the Report SlaveID request.

---

requestUserFunction

```
TypeScript
requestUserFunction(functionID: number, data: number[] | Uint8Array): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`functionID`

8-bit function ID.

`data`

Integer array of function parameters. Each item is treated as 8-bit unsigned value.

Description

Construct the user-defined message to the device.

Example

```JavaScript
var message = builder.requestUserFunction(15, [ 1, 2, 3, 4, 5, 6, 7 ]); // User function 15
with 7 parameters
```

---

requestWriteFileRecord

```TypeScript
requestWriteFileRecord(requests: FileRecord[]): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`requests`

An array of file records. The function throws an exception if you specify incorrect values for the parameter. The function also throws an exception if there is too many values, as well as if the total length of the packet exceeds 255 bytes.

Description

Construct the Write File Record request.

Example

```JavaScript
var message = builder.requestWriteFileRecord([
  { referencyType: 1, fileNumber: 10, recordNumber: 20, records: [ 255 ] },
  { referencyType: 1, fileNumber: 10, recordNumber: 20, records: [ 255, 255 ] },
  { referencyType: 1, fileNumber: 11, recordNumber: 21, records: [ 127, 127, 127] }
  ]);
```

---

requestWriteMultipleCoils

```TypeScript
requestWriteMultipleCoils(startingAddress: number, data: boolean[]): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`startingAddress`

Starting address (number from 0 to 65535).

`data`

An array of COIL values.

Description

Construct the Write Multiple Coils request.

---

requestWriteMultipleRegisters

```
TypeScript
requestWriteMultipleRegisters(startingAddress: number, data: number[] | Uint16Array):
Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`startingAddress`

Starting address (number from 0 to 65535).

`data`

An array of register values. Each value must be between 0 and 65535.

Description

Construct the Write Multiple Registers request.

---

requestWriteSingleCoil

```
TypeScript
requestWriteSingleCoil(outputAddress: number, outputValue: number): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`outputAddress`

16-bit coil address (0 to 65535).

`outputValue`

Output value is used to set the coil state to FALSE/TRUE values. Only 0 to 65535 (0x0000 to 0xFFFF) values are accepted. However, the documentation allows only 0 and 65280 (0x0000 and 0xFF00) values to be used (see the MODBUS protocol documentation) to switch the coil state to FALSE/TRUE.

Description

Construct the Write Single Coil request .

Example

```JavaScript
var m1 = builder.requestWriteSingleCoil(0, 65280); // set the 1 (TRUE) value to coil 0
var m2 = builder.requestWriteSingleCoil(1, 0); // set the 0 (FALSE) value to coil 1
var m3 = builder.requestWriteSingleCoil(2, 15);  // WARNING: the value 15 is undefined by
standard.
// coil #2 will have unknown value now (FALSE or TRUE)
```

---

requestWriteSingleRegister

```TypeScript
requestWriteSingleRegister(registerAddress: number, registerValue: number): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`registerAddress`

Register address (0 to 65535). In some implementations device interprets this value as an offset — this value is added to default MODBUS base register address (see MODBUS protocol documentation) 40000. Specify the 0 value for this parameter to write register with 40000 address.

`registerValue`

Value to be set. The parameter can take values from 0 to 65535 (0x0000 to 0xFFFF).

Description

Construct the Write Single Register request.

---

responseDiagnostics

```TypeScript
responseDiagnostics(subFunction: number, data: number [] | Uint8Array): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`subFunction`

The sub-function index. It can be a number from 0 (0x0000) to 65535 (0xFFFF).

`data`

Byte array with diagnostics response data.

Description

Send the Diagnostics response.

The function throws an exception if you specify incorrect values for the `data` parameter. The function also throws an exception if the total length of the packet exceeds 255 bytes.

---

responseGetCommEventCounter

```TypeScript
responseGetCommEventCounter(status: number, eventCount: number): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`status`

Status value. It can be a number from 0 (0x0000) to 65535 (0xFFFF).

`eventCount`

Event count value. It can be a number from 0 (0x0000) to 65535 (0xFFFF).

Description

Construct the Get Comm Event Counter response.

---

responseGetCommEventLog

```TypeScript
responseGetCommEventLog(status: number, events: number [] | Uint8Array): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`status`

Status value. It can be a number from 0 (0x0000) to 65535 (0xFFFF). messageCount: numberEvent count value. It can be a number from 0 (0x0000) to 65535 (0xFFFF).

`events`

Integer array of 8-bit event values.

Description

Construct the Get Comm Event Log response.

responseMaskWriteRegister

```TypeScript
responseMaskWriteRegister(referenceAddress: number, andMask: number, orMask: number):
Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`referenceAddress`

Reference address. It can be a number from 0 to 65535.

`andMask`

And Mask is used for bitwise AND operation on the selected register address. It can be a number from 0 to 65535.

`orMask`

And Mask is used for bitwise OR operation on the selected register address. It can be a number from 0 to 65535.

Description

Construct the Mask Write Register response.

responseReadCoilStatus

```TypeScript
responseReadCoilStatus(items: number [] | Uint8Array): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`items`

Integer array of 8-bit coil values.

Description

Construct the Read Coil Status response.

responseReadDiscreteInputs

```
TypeScript
responseReadDiscreteInputs(items: number [] | Uint8Array): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`items`

Integer array of 8-bit coils values.

Description

Construct the Read Discrete Inputs response.

---

responseReadExceptionStatus

```
TypeScript
responseReadExceptionStatus(outputData: number): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`outputData`

The output value. It can be a number from 0 (0x00) to 255 (0xFF).

Description

Construct the Read Exception Status response.

---

responseReadFIFOQueue

```
TypeScript
responseReadFIFOQueue(registers: number [] | Uint16Array): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`registers`

Integer array of 16-bit register values.

Description

Send the Read FIFO Queue response.

The function throws an exception if you specify incorrect values for the `registers` parameter. The function also throws an exception if the total length of the packet exceeds 255 bytes.

---

responseReadFileRecord

```
TypeScript
responseReadFileRecord(requests: ReadFileResponse[]): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`requests`

An array of file response records. The function throws an exception if you specify incorrect values for the parameter. The function also throws an exception if there are too many values, as well as if the total length of the packet exceeds 255 bytes.

Description

Construct the Read File Record response.

Example

```
JavaScript
var message = builder.responseReadFileRecord([
  { referenceType: 6, data: [ 255 ] },
  { referenceType: 6, data: [ 65535, 65535 ] },
  { referenceType: 6, data: [ 120 ] }
  ]);
```

---

responseReadHoldingRegisters

```
TypeScript
responseReadHoldingRegisters(registers: number [] | Uint16Array): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`registers`

Register values array. Each item is treated as unsigned 16-bit integer.

Description

Construct the Read Holding Registers response.

The function throws an exception if you specify incorrect values for the `registers` parameter. The function also throws an exception if the total length of the packet exceeds 255 bytes.

---

responseReadInputRegisters

```
TypeScript
responseReadInputRegisters(registers: number [] | Uint16Array): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`registers`

Array with register values. Each value is treated as unsigned 16-bit integer.

Description

Construct the Read Input Registers response.

The function throws an exception if you specify incorrect values for the `registers` parameter. The function also throws an exception if the total length of the packet exceeds 255 bytes.

---

responseReadWriteRegisters

```
TypeScript
responseReadWriteRegisters(registers: number [] | Uint16Array): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`registers`

Array of 16-bit register values.

Description

Construct the Read Write Registers response.

---

responseReportSlaveID

```
TypeScript
responseReportSlaveID(runIndicatorStatus: number,
    slaveIds: number [] | Uint8Array,
    data: number [] | Uint8Array): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`runIndicatorStatus`

RunIndicatorStatus value. It can be a value from 0 to 255.

`slaveIds`

An array of slave identifiers.

`data`

Additional data to send.

Description

Construct the Report SlaveID response.

The function throws an exception if you specify incorrect values for the `requests` parameter. The function also throws an exception if there is too many or too little values, as well as if the total length of the packet exceeds 255 bytes.

Example

```
JavaScript
// send 1 byte (255 in this example) as SlaveId field data
// and 2 bytes (255,100 in this example) as AdditionalData field data
// RunIndicatorStatus field is 0
var message = builder.responseReportSlaveID(0, [ 255 ], [ 255, 100 ]);
```

---

responseWriteFileRecord

```
TypeScript
responseWriteFileRecord(requests: FileRecord[]): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`requests`

An array of file records. The function throws an exception if you specify incorrect values for the parameter. The function also throws an exception if there is too many values, as well as if the total length of the packet exceeds 255 bytes.

Description

Construct the Write File Record response.

Example

```JavaScript
var message = builder.responseWriteFileRecord([
  { referenceType: 1, fileNumber: 2, recordNumber: 1, records: [ 65535 ] }, // first request
(with 1 record data value)
  { referenceType: 1, fileNumber: 40000, recordNumber: 2, records: [ 65535, 65535, 65535 ] },
// second request (with 3 record data values)
  { referenceType: 1, fileNumber: 2, recordNumber: 3, records: [ 120 ] }  // third request
(with 1 record data value)
  ]);
```

---

responseWriteMultipleCoils

```TypeScript
responseWriteMultipleCoils(startingAddress: number, quantityOfOutputs: number): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`startingAddress`

Starting address value. It can be a number from 0 (0x0000) to 65535 (0xFFFF).

`quantityOfOutputs`

Outputs count value. It can be a number from 0 (0x0001) to 1968 (0x07B0).

Description

Construct the Write Multiple Coils response.

---

responseWriteMultipleRegisters

```TypeScript
responseWriteMultipleRegisters(startingAddress: number, quantityOfRegisters: number):
Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`startingAddress`

Starting address value. It can be a number from 0 (0x0000) to 65535 (0xFFFF).

`quantityOfRegisters`

Event count value. It can be a number from 1 (0x0001) to 123 (0x007B).

Description

Construct the Write Multiple Registers response.

---

responseWriteSingleCoil

```
TypeScript
responseWriteSingleCoil(outputAddress: number, outputValue: number): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`outputAddress`

The address of a coil. It can be a number from 0 (0x0000) to 65535 (0xFFFF).

`outputValue`

The output value. It can be a number from 0 (0x0000) to 65280 (0xFF00). A value of 65280 (0xFF00) requests the output to be ON (TRUE). A value of 0 (0x0000) requests it to be OFF (FALSE).

Description

Construct the Write Single Coil response.

---

responseWriteSingleRegister

```
TypeScript
responseWriteSingleRegister(outRegisterAddress: number, outValue: number): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`outRegisterAddress`

The register address. It can be a number from 0 (0x0000) to 65535 (0xFFFF).

`outValue`

The output value. It can be a number from 0 (0x0000) to 65535 (0xFFFF).

Description

Construct the Write Single Register response.

ReadFileRequest Interface

```typescript
interface ReadFileRequest {
    // Properties
    referenceType: number;
    fileNumber: number;
    recordNumber: number;
    registerLength: number;
}
```

```csharp
// This interface is not available in managed environment
```

```cpp
// This interface is not available in native environment
```

ReadFileRequest Properties

---

referenceType

```typescript
referenceType: number;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Reference type.

---

fileNumber

```typescript
fileNumber: number;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

File number.

---

recordNumber

```
TypeScript
recordNumber: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Record number.

---

registerLength

```
TypeScript
registerLength: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Register length.

ReadFileResponse Interface

```
TypeScript
interface ReadFileResponse {
    // Properties
    referenceType: number;
    data: number [] | Uint16Array;
}
```

```
C#
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

ReadFileResponse Properties

---

referenceType

```
TypeScript
referenceType: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Reference type.

---

data

```
TypeScript
data: number [] | Uint16Array;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Record data.

FileRecord Interface

```
TypeScript
interface FileRecord {
    // Properties
    referenceType: number;
    fileNumber: number;
    recordNumber: number;
    records: number [] | Uint16Array;
}
```

```
C#
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

FileRecord Properties

---

referenceType

```
TypeScript
referenceType: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Reference type.

---

fileNumber

```
TypeScript
fileNumber: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

File number.

---

recordNumber

```
TypeScript
recordNumber: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Record number.

---

records

```
TypeScript
records: number [] | Uint16Array;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Data payload.

Remote Connection Manager Object

Remote Connection Manager Object allows you to connect to remote server, disconnect from remote server and enumerate active connections. It is exposed by the global object `remote` and implements the IRemoteHost interface.

Use the IRemoteHost.connectServer method to establish a remote connection. Use the IRemoteHost.disconnectServer to close the remote connection and query the IRemoteHost.connections property for a list of existing connections.

Events

Subscribe to IRemoteHost.connected or IRemoteHost.disconnected events to get notifications of connection and disconnection events.

IRemoteHost Interface

Description

This interface consists of methods, properties and events supported by the Remote Connection Manager Object.

This interface is exposed by the global object `remote`.

Declaration

TypeScript
```
interface IRemoteHost {
    // Properties
    readonly connections: string[];

    // Methods
    connectServer(serverName: string): boolean;
    disconnectServer(serverName: string): boolean;

    // Events
    connected(handler: (serverName: string) => void): number;
    connected(eventId: number): void;
    disconnected(handler: (serverName: string) => void): number;
    disconnected(eventId: number): void;
}
```

C#
```
// This interface is not available in managed environment
```

C++
```
// This interface is not available in native environment
```

IRemoteHost Properties

---

connections

TypeScript
```
readonly connections: string[];
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Returns the array of strings representing connected computer names.

Example

Connect to a remote server and print the number of connections:

```JavaScript
if (remote.connectServer("remote_server_name"))
    alert("Number of connections: " + remote.connections.length);
```

IRemoteHost Methods

---

connectServer

```TypeScript
connectServer(serverName: string): boolean;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`serverName`

Server name.

Return Value

Returns `true` if connection was successfully made, and `false` otherwise.

Description

Establishes a remote connection. Pass a resolvable computer name (DNS name, NETBIOS name, IP address and so on) of a remote computer.

Example

Connect to a remote server and print the number of connections:

```JavaScript
if (remote.connectServer("remote_server_name"))
    alert("Number of connections: " + remote.connections.length);
```

---

disconnectServer

```TypeScript
disconnectServer(serverName: string): boolean;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`serverName`

Server name.

Return Value

Returns `true` if connection was successfully disconnected, and `false` otherwise.

Description

Closes a remote connection. Pass a resolvable computer name (DNS name, NETBIOS name, IP address and so on) of a remote computer.

Example

Disconnect a given connection:

```JavaScript
remote.disconnectServer("remote_server_name");
```

IRemoteHost Events

---

connected

```TypeScript
connected(handler: (serverName: string) => void): number;
connected(eventId: number): void;
```

Parameters

`serverName`

Name of a remote computer.

Description

This event is fired when new remote connection is established.

First overload is used to bind new handler to the event. It returns a numeric `eventId` which then may be passed to second overload to unbind a handler.

Example

Subscribing to connected and disconnected events:

```TypeScript
remote.connected(serverName=>alert("Connected: " + serverName + ". Total connections: " +
remote.connections.length));
remote.disconnected(serverName=>alert("Disconnected: " + serverName + ". Total connections: "
+ remote.connections.length));
```

---

disconnected

```TypeScript
disconnected(handler: (serverName: string) => void): number;
disconnected(eventId: number): void;
```

Parameters

`serverName`

Name of a remote computer.

Description

This event is fired when existing remote connection is closed.

First overload is used to bind new handler to the event. It returns a numeric `eventId` which then may be passed to second overload to unbind a handler.

Example

Subscribing to connected and disconnected events:

```TypeScript
remote.connected(serverName=>alert("Connected: " + serverName + ". Total connections: " +
remote.connections.length));
remote.disconnected(serverName=>alert("Disconnected: " + serverName + ". Total connections: "
+ remote.connections.length));
```

Bridge Manager Object

Bridge Manager Object allows you to create, enumerate and persist serial bridges. It is exposed by the global object `bridge` and implements the IBridgeHost interface.

Use the IBridgeHost.create method to create new serial bridge. Use the IBridgeHost.bridges property to get all configured serial bridges. Use IBridgeHost.saveConfiguration and IBridgeHost.loadConfiguration methods to save and load current configuration.

Use one of the following syntaxes to start serial bridge monitoring session:

```TypeScript
var b = bridge.create("COM1", "COM2");

// 1. Use generic overload and match generic device name
var session = monitoring.createSession("Communication Port (COM1) <-> Communication Port
(COM2)");

// 2. Use specific bridge overload and match device name
var session = monitoring.createSession({ bridge: b.name });

// 3. Use specific bridge overload and pass bridge object directly
var session = monitoring.createSession({ bridge: b });
```

IBridgeHost Interface

Description

This interface consists of methods and properties supported by the Bridge Manager Object.

This interface is exposed by the global object `bridge`.

Declaration

TypeScript
```
interface IBridgeHost {
    // Properties
    bridges: IBridge[];

    // Methods
    create(firstDevice: string,
        secondDevice: string,
        firstDeviceConfig?: IDeviceConfig,
        secondDeviceConfig?: IDeviceConfig): IBridge;
    saveConfiguration(name?: string): boolean;
    loadConfiguration(name?: string): boolean;
}
```

C#
```
// This interface is not available in managed environment
```

C++
```
// This interface is not available in native environment
```

IBridgeHost Properties

---

bridges

TypeScript
```
bridges: IBridge[];
```

C#
```
// This property is not available in managed environment
```

C++
```
// This property is not available in native environment
```

Description

Returns the array of all created serial bridges.

IBridgeHost Methods

---

create

```TypeScript
create(firstDevice: string,
    secondDevice: string,
    firstDeviceConfig?: IDeviceConfig,
    secondDeviceConfig?: IDeviceConfig): IBridge;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`firstDevice`

First serial device name.

`secondDevice`

Second serial device name.

`firstDeviceConfig`

First serial device configuration object. If omitted, system default is used.

`secondDeviceConfig`

Second serial device configuration object. If omitted, bridge will use the same configuration for both devices.

Return Value

Returns created serial bridge object.

Description

Creates new serial bridge.

Example

Create new serial bridge:

```JavaScript
// Create bridge between COM1 and COM2 with baud rate 115200, 8 byte size and hardware flow
control
// All other parameters are taken from COM1's defaults
var b = bridge.create("COM1", "COM2", {
    baudRate: 115200,
    dataBits: 8,
    flowControl: flowControl.hardware
});
```

---

saveConfiguration

```TypeScript
saveConfiguration(name?: string): boolean;
```

```C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`name`

Optional configuration name. If omitted or passed empty string, saves default configuration. Default configuration is automatically loaded on Device Monitoring Studio startup.

Return Value

Returns `true` if configuration has been successfully saved, `false` otherwise.

Description

Saves current serial bridge configuration.

---

loadConfiguration

```
TypeScript
loadConfiguration(name?: string): boolean;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`name`

Optional configuration name. If omitted or passed empty string, loads default configuration.

Return Value

Returns `true` if configuration has been successfully loaded, `false` otherwise.

Description

Loads serial bridge configuration.

IBridge IDeviceConfig

Bridge Object

Bridge Object represents the created serial bridge. You get a reference to the bridge object either by calling IBridgeHost.create method or by enumerating the IBridgeHost.bridges property.

Query the IBridge.firstDevice or IBridge.secondDevice properties to get or change bridge device properties, query the IBridge.name property to get auto-assigned bridge name or use the IBridge.destroy method to delete bridge.

IBridge Interface

Description

This interface consists of methods and properties supported by the Bridge Object.

Declaration

```
TypeScript
interface IBridge {
    // Properties
    firstDevice: IDeviceConfig;
    secondDevice: IDeviceConfig;
    name: string;

    // Methods
    destroy(): void;
}
```

```
C#
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

IBridge Properties

---

firstDevice

```
TypeScript
firstDevice: IDeviceConfig;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Returns the reference to bridge's first device configuration object.

---

secondDevice

```
TypeScript
secondDevice: IDeviceConfig;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Returns the reference to bridge's second device configuration object.

---

name

```
TypeScript
name: string;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Returns the auto-assigned bridge name.

IBridge Methods

---

destroy

```
TypeScript
destroy(): void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Description

Deletes the serial bridge.

IDeviceConfig

File Manager Object

File Manager object provides a number of methods to access the file system. It implements the IFileManager interface and is available through a global object `fileManager`.

Use IFileManager.createFile method to create or open a file.

Use IFileManager.deleteFile to delete an existing file.

IFileManager.createFolder and IFileManager.deleteFolder may be used to create or delete folder, respectively.

Call IFileManager.enumFiles to enumerate files in a folder.

Reference

IFileManager Interface

Description

This interface is implemented by File Manager Object. It provides basic methods to work with a file system, like opening or creating files and folders, deleting files and folders and enumerating the contents of a folder.

Declaration

```TypeScript
interface IFileManager {

    // Methods
    createFile(path: string,
        openMode: File.OpenMode,
        access: File.Access,
        share?: File.Share): IFile;
    deleteFile(path: string): void;
    enumFiles(folder: string, mask?: string): Promise<string[]>;
    copyFile(source: string, destination: string, overwrite?: boolean): Promise<void>;
    moveFile(source: string, destination: string, overwrite?: boolean): Promise<void>;
    createFolder(path: string): void;
    deleteFolder(path: string): void;
}
```

```C#
// This interface is not available in managed environment
```

```C++
// This interface is not available in native environment
```

IFileManager Methods

---

createFile

```TypeScript
createFile(path: string,
    openMode: File.OpenMode,
    access: File.Access,
    share?: File.Share): IFile;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`path`

Full path to the file being opened or created.

`openMode`

File opening mode. See File.OpenMode topic for more information.

`access`

File access mode. A file may be opened for reading, writing or both reading and writing.

`share`

File sharing mode. Tells the file system how it should handle other process attempts to open a file.

If omitted, equals to `File.Share.Exclusive`.

Return Value

An opened file object.

Description

Creates or opens a file.

Example

Open an existing file for reading

```TypeScript
var file = fileManager.createFile("c:\\temp\\file.txt", File.OpenMode.OpenExisting,
File.Access.Read, File.Share.Read);
```

---

deleteFile

```TypeScript
deleteFile(path: string): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`path`

Full path to the file to delete.

Description

Deletes a given file.

---

enumFiles

```TypeScript
enumFiles(folder: string, mask?: string): Promise<string[]>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`folder`

Full path to the folder the caller wants to enumerate.

`mask`

An optional mask to match files during enumeration. If omitted, equals to "*".

Description

Enumerate files in a given folder. The method executes asynchronously and returns a list of file names that match a given mask.

Example

Enumerating files in a folder

```
TypeScript
var files = await fileManager.enumFiles("c:\\temp", "*.txt");
```

---

copyFile

```
TypeScript
copyFile(source: string, destination: string, overwrite?: boolean): Promise<void>;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`source`

Full path to the source file.

`destination`

Full path to the destination file.

`overwrite`

An optional parameter that tells if the destination file should be overridden if it exists. If omitted, defaults to `false`.

Description

Copies a source file to destination. The method executes asynchronously.

Example

Copying a file

```
TypeScript
await fileManager.copyFile(source, destination);
```

---

moveFile

```
TypeScript
moveFile(source: string, destination: string, overwrite?: boolean): Promise<void>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`source`

Full path to the source file.

`destination`

Full path to the destination file.

`overwrite`

An optional parameter that tells if the destination file should be overridden if it exists. If omitted, defaults to `false`.

Description

Moves a source file to destination or renames a file. The method executes asynchronously.

Example

Renaming a file

```TypeScript
await fileManager.moveFile(source, destination);
```

---

createFolder

```TypeScript
createFolder(path: string): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`path`

Full path to the folder being created.

Description

Creates a folder. If one or more intermediate folders in a given path do not exist, they are also created by this function.

---

deleteFolder

```
TypeScript
deleteFolder(path: string): void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`path`

Full path to the folder being deleted.

Description

Deletes a given folder. A folder must be empty to be successfully deleted.

File Namespace

File.OpenMode Enumeration

Description

File opening mode constants. See IFileManager.createFile for more information.

Symbol	Value	Description
`CreateNew`	`1`	Creates a new file, only if it does not already exist. If the specified file exists, the function fails with "File Exists" exception. If the specified file does not exist and is a valid path to a writable location, a new file is created.
`CreateAlways`	`2`	Creates a new file, always. If the specified file exists and is writable, the function overwrites the file. If the specified file does not exist and is a valid path, a new file is created.
`OpenExisting`	`3`	Opens a file or device, only if it exists. If the specified file does not exist, the function fails with "File Not Found" exception.
`OpenAlways`	`4`	Opens a file, always. If the specified file does not exist and is a valid path to a writable location, the function creates a file.
`TruncateExisting`	`5`	Opens a file and truncates it so that its size is zero bytes, only if it exists. If the specified file does not exist, the function fails with "File Not Found" exception. The caller must open the file with the `File.Access.Write` access specifier.

File.Access Enumeration

Description

File access rights constants. See IFileManager.createFile function for more information.

Symbol	Value	Description
`Read`	`1`	Opens a file for reading.
`Write`	`2`	Opens a file for writing.
`ReadWrite`	`3`	Opens a file both for reading and writing.

File.Share Enumeration

Description

File sharing rights. See IFileManager.createFile function for more information.

Symbol	Value	Description
`Exclusive`	`0`	Opens a file for exclusive access.
`Read`	`1`	Allows other processes to read from a file.
`Write`	`2`	Allows other processes to write to a file.
`Delete`	`4`	Allows other processes to delete a file.

File Object

File object implements the IFile interface. File objects are created using the IFileManager.createFile method.

File objects provide methods to read and write file data, query file size and query and update the current file position.

IFile Interface

Description

This interface is implemented by a File Object.

Declaration

TypeScript
```
interface IFile {
    // Properties
    currentPosition: number;
    readonly size: number;
    readonly isOpen: boolean;

    // Methods
    read(size: number, position?: number): Promise<Uint8Array>;
    write(data: number[] | Uint8Array | DataView | ArrayBuffer, position?: number):
Promise<number>;
    setEnd(): void;
    close(): void;
}
```

C#
```
// This interface is not available in managed environment
```

C++
```
// This interface is not available in native environment
```

IFile Properties

currentPosition

```TypeScript
currentPosition: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

This property represents the current file's position.

size

```TypeScript
readonly size: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

This property holds the current file's size.

isOpen

```TypeScript
readonly isOpen: boolean;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

This property evaluates to `false` after calling the IFile.close method.

@returns `false` after calling IFile.close method, `true` otherwise.

IFile Methods

read

```
TypeScript
read(size: number, position?: number): Promise<Uint8Array>;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`size`

The number of bytes to read.

`position`

Optional starting offset for the operation. If omitted, the file's current position is used (and later updated).

Return Value

Data read from a file.

Description

Reads data from a file. This method executes asynchronously.

Example

Reading from a file

```
TypeScript
var data = await file.read(4096);
```

---

write

```
TypeScript
write(data: number[] | Uint8Array | DataView | ArrayBuffer, position?: number):
Promise<number>;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`data`

The data to be written.

`position`

Optional starting offset for the operation. If omitted, the file's current position is used (and later updated).

Return Value

The number of bytes written.

Description

Writes data to a file. This method executes asynchronously.

Example

Writing to a file

```TypeScript
// Copy first 4KB of file to second 4KB

var data = await file.read(4096, 0);
await file.write(data, 4096);
```

---

setEnd

```TypeScript
setEnd(): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Description

Changes the size of a file to be the same as IFile.currentPosition.

---

close

```TypeScript
close(): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Description

Closes the file. File object may not be used after calling this method. Only IFile.isOpen property may safely be used after calling this method.

HID Manager Object

HID Manager object provides a scripting access to HID Send module. It is available as a global variable `hid` and implements the IHidManager interface.

IHIDManager.devices property returns an array of supported HID devices. Call the IHIDManager.createSession method to create new HID Send session object and use IHIDManager.sessions property to get an array of all currently running sessions.

IHIDManager.closeAllSessions method may be called to stop all running sessions.

IHIDManager Interface

Description

This interface is implemented by HID Manager.

Declaration

```
TypeScript
interface IHIDManager {
    // Properties
    readonly devices: IHIDDevice[];
    sessions: IHIDSession[];

    // Methods
    createSession(device: IHIDDevice): IHIDSession;
    createSession(deviceKey: string): IHIDSession;
    createSession(vendorId: number,
        productId: number,
        searchOptions?: { serialNumber?: string; usage?: number; usagePage?: number }):
IHIDSession;
    closeAllSessions(): void;
}
```

```
C#
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

IHIDManager Properties

---

devices

```
TypeScript
readonly devices: IHIDDevice[];
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Returns the array of HID device objects.

Example

Obtain a list of detected HID devices and their information:

```JavaScript
var devices = hid.devices;
for (var i = 0; i < devices.length; ++i)
    alert(devices[i].vendorId + "." + devices[i].productId);
```

sessions

```TypeScript
sessions: IHIDSession[];
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Returns the array of HID session objects.

Example

Obtain a list of HID sessions and print their corresponding device information:

```JavaScript
var sessions = hid.sessions;
for (var i = 0; i < sessions.length; ++i)
    alert(devices[i].vendorId + "." + devices[i].productId);
```

IHIDManager Methods

createSession

```TypeScript
createSession(device: IHIDDevice): IHIDSession;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`device`

A device object. Use the IHIDManager.devices property to get a reference to a device object.

Return Value

Returns the reference to created session object.

Description

Creates new HID session.

Example

Create new HID session:

```JavaScript
var session1 = hid.createSession(hid.devices[0]);
```

---

createSession

```TypeScript
createSession(deviceKey: string): IHIDSession;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`deviceKey`

A HID device identifier (also known as device key).

Return Value

Returns the reference to created session object.

Description

Creates new HID Session.

---

createSession

```TypeScript
createSession(vendorId: number,
    productId: number,
    searchOptions?: { serialNumber?: string; usage?: number; usagePage?: number }):
IHIDSession;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`vendorId`

A HID device vendor ID.

`productId`

A HID device product ID.

`searchOptions`

Optional search parameters. Use it when there can be several devices of the same type connected to a computer. You can specify device serial number, usage or usage page.

Return Value

Returns the reference to created session object.

Description

Creates new HID Session.

Example

Create new HID session:

```JavaScript
var session = hid.createSession(0x1234, 0x5678, { serialNumber: "XXX0001" });
```

---

closeAllSessions

```TypeScript
closeAllSessions(): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Description

Stops and closes all running HID sessions.

HID Device Object

HID Device represents detected HID device. Device object implementsIHIDDevice interface and allows you to query various device properties.

IHIDDevice Interface

Description

This interface is implemented by HID Device Object. All properties are read-only.

Declaration

```typescript
interface IHIDDevice {
    // Properties
    deviceKey: string;
    vendorId: number;
    productId: number;
    serialNumber: string;
    releaseNumber: number;
    manufacturer: string;
    product: string;
    interfaceNumber: number;
    caps: IHIDCaps;
}
```

```csharp
// This interface is not available in managed environment
```

```cpp
// This interface is not available in native environment
```

IHIDDevice Properties

---

deviceKey

```typescript
deviceKey: string;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

HID device identifier (device key).

---

vendorId

```typescript
vendorId: number;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

HID device vendor identifier.

---

productId

```TypeScript
productId: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

HID device product identifier.

---

serialNumber

```TypeScript
serialNumber: string;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

HID device serial number.

---

releaseNumber

```TypeScript
releaseNumber: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

HID device release number (also known as Manufacturer Release Number).

---

manufacturer

```TypeScript
manufacturer: string;
```

```C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

HID device manufacturer.

---

product

```
TypeScript
product: string;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

HID device product name.

---

interfaceNumber

```
TypeScript
interfaceNumber: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

HID device interface number.

---

caps

```
TypeScript
caps: IHIDCaps;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

HID device usage page.

HID Session Object

HID Session object represents a configured HID session. Please see methods and properties of the IHIDSession interface for capabilities and actions of a session object.

You obtain a reference to a session object by calling IHIDManager.createSession method.

Reference

HID Namespace

HID.ReportType Enumeration

Description

HID report types.

Symbol	Value	Description
`Input`	`0`	Indicates an input report.
`Output`	`1`	Indicates an output report.
`Feature`	`2`	Indicates a feature report.

IHIDSession Interface

Description

This interface is implemented by HID Session Object.

Declaration

```TypeScript
interface IHIDSession {
    // Properties
    vendorId: number;
    productId: number;
    device: IHIDDevice;
    inputBuffersCount: number;

    // Methods
    start(): void;
    stop(): void;
    setFeature(reportId: number, v: number[] | ArrayBuffer | DataView | Uint8Array): void;
    getFeature(reportId: number): Uint8Array;
    send(byte: number): Promise<void>;
    send(bytes: number [] | Uint8Array | Uint16Array | Uint32Array | ArrayBuffer | DataView):
Promise<void>;
    send(text: string): Promise<void>;
    receive(): Promise<Uint8Array>;
    getReport(reportId: number): Uint8Array;
    setReport(reportId: number, v: number[] | ArrayBuffer | DataView | Uint8Array): void;
    getLinkCollectionNodes(): IHIDNode[];
    getValueCaps(reportType: HID.ReportType): IHIDValueCaps[];
    getSpecificValueCaps(reportType: HID.ReportType, usagePage: number, usage: number,
linkCollection: number): IHIDValueCaps[];
    getButtonCaps(reportType: HID.ReportType): IHIDButtonCaps[];
    getSpecificButtonCaps(reportType: HID.ReportType, usagePage: number, usage: number,
linkCollection: number): IHIDButtonCaps[];
    createBuilder(): IHIDBuilder;
    createParser(): IHIDParser;
}
```

```C#
// This interface is not available in managed environment
```

```C++
// This interface is not available in native environment
```

IHIDSession Properties

---

vendorId

```TypeScript
vendorId: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

HID device vendor identifier.

---

productId

```TypeScript
productId: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

HID device product identifier.

---

device

```TypeScript
device: IHIDDevice;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Reference to the HID device object.

---

inputBuffersCount

```TypeScript
inputBuffersCount: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Number of input buffers.

IHIDSession Methods

---

start

```TypeScript
start(): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Description

Starts a HID session.

---

stop

```TypeScript
stop(): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Description

Stops a HID session.

---

setFeature

```TypeScript
setFeature(reportId: number, v: number[] | ArrayBuffer | DataView | Uint8Array): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`reportId`

Feature report ID.

`v`

Raw feature report bytes.

Description

Sends feature report.

---

getFeature

```TypeScript
getFeature(reportId: number): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`reportId`

Feature report id.

Return Value

Raw feature report bytes.

Description

Receive feature report.

---

send

```
TypeScript
send(byte: number): Promise<void>;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`byte`

Single byte to send.

Description

Send data to a HID session.

---

send

```
TypeScript
send(bytes: number [] | Uint8Array | Uint16Array | Uint32Array | ArrayBuffer | DataView):
Promise<void>;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`bytes`

A JavaScript array of integer numbers to be sent to the session.

Description

Send data to a HID session.

---

send

```TypeScript
send(text: string): Promise<void>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`text`

A text string to send (ASCII encoding).

Description

Send data to a HID session.

---

receive

```TypeScript
receive(): Promise<Uint8Array>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Description

Receive (read) data from the session. This function returns a promise that produces a byte array when ready.

---

getReport

```TypeScript
getReport(reportId: number): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`reportId`

Feature report ID.

Return Value

Raw input report bytes.

Description

Receive input report.

---

setReport

```
TypeScript
setReport(reportId: number, v: number[] | ArrayBuffer | DataView | Uint8Array): void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`reportId`

Output report ID.

`v`

Raw output report bytes.

Description

Sends output report.

---

getLinkCollectionNodes

```
TypeScript
getLinkCollectionNodes(): IHIDNode[];
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Description

Returns a top-level collection's link collection array.

---

getValueCaps

```
TypeScript
getValueCaps(reportType: HID.ReportType): IHIDValueCaps[];
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

Description

Returns a value capability array that describes all the HID control values in a top-level collection for a specified type of HID report.

---

getSpecificValueCaps

```
TypeScript
getSpecificValueCaps(reportType: HID.ReportType, usagePage: number, usage: number,
linkCollection: number): IHIDValueCaps[];
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

`usagePage`

Specifies a usage page as a search criteria. If `usagePage` is nonzero, only buttons that specify this usage page are returned.

`usage`

Specifies a HID usage as a search criteria. If `usage` is nonzero, only buttons that specify this usage will be returned.

`linkCollection`

Specifies a link collection as a search criteria. If `linkCollection` is nonzero, only buttons that are part of this link collection are returned.

Description

Returns a value capability array that describes all HID control values that meet a specified selection criteria.

---

getButtonCaps

```TypeScript
getButtonCaps(reportType: HID.ReportType): IHIDButtonCaps[];
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

Description

Returns a button capability array that describes all the HID control buttons in a top-level collection for a specified type of HID report.

---

getSpecificButtonCaps

```TypeScript
getSpecificButtonCaps(reportType: HID.ReportType, usagePage: number, usage: number,
linkCollection: number): IHIDButtonCaps[];
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

`usagePage`

Specifies a usage page as a search criteria. If `usagePage` is nonzero, only buttons that specify this usage page are returned.

`usage`

Specifies a HID usage as a search criteria. If `usage` is nonzero, only buttons that specify this usage will be returned.

`linkCollection`

Specifies a link collection as a search criteria. If `linkCollection` is nonzero, only buttons that are part of this link collection are returned.

Description

Returns a button capability array that describes all HID control buttons in a top-level collection that meet a specified selection criteria.

---

createBuilder

```
TypeScript
createBuilder(): IHIDBuilder;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Description

Create a HID report builder object. This object implementsIHIDBuilder interface and allows you to construct HID reports given the required parameters.

---

createParser

```
TypeScript
createParser(): IHIDParser;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Description

Create a HID report parser object. This object implementsIHIDParser interface and allows you to parse reports received from the device.

IHIDDevice

IHIDParser Interface

Description

This interface is implemented by HID Session object and allows you to parse the HID report received from the device.

Declaration

```TypeScript
interface IHIDParser {

    // Methods
    getData(reportType: HID.ReportType, reportData: Uint8Array): IHIDData[];
    getUsages(reportType: HID.ReportType,
        usagePage: number,
        linkCollection: number,
        reportData: Uint8Array): number[];
    getUsagesEx(reportType: HID.ReportType, linkCollection: number, reportData: Uint8Array):
IHIDUsageAndPage[];
    getButtons(reportType: HID.ReportType, usagePage: number, linkCollection: number,
reportData: Uint8Array): number[];
    getButtonsEx(reportType: HID.ReportType, linkCollection: number, reportData: Uint8Array):
IHIDUsageAndPage[];
    getUsageValue(reportType: HID.ReportType, usagePage: number, usage: number, linkCollection:
number): number;
    getScaledUsageValue(reportType: HID.ReportType,
        usagePage: number,
        usage: number,
        linkCollection: number,
        reportData: Uint8Array): number;
    getUsageValueArray(reportType: HID.ReportType,
        usagePage: number,
        usage: number,
        linkCollection: number,
        reportData: Uint8Array): Uint8Array;
}
```

```C#
// This interface is not available in managed environment
```

```C++
// This interface is not available in native environment
```

IHIDParser Methods

---

getData

```TypeScript
getData(reportType: HID.ReportType, reportData: Uint8Array): IHIDData[];
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

`reportData`

Raw HID report bytes.

Description

Returns, for a specified report, an array of data objects that identify the data indices of all HID control buttons that are currently set to ON (1), and the data indices and data associated with all HID control

values.

---

getUsages

```
TypeScript
getUsages(reportType: HID.ReportType,
    usagePage: number,
    linkCollection: number,
    reportData: Uint8Array): number[];
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

`usagePage`

Specifies the usage page of the button usages. The method only returns information about buttons on this usage page.

`linkCollection`

Specifies the link collection of the button usages. If `linkCollection` is nonzero, the routine only returns information about the buttons that this link collection contains; otherwise, if `linkCollection` is zero, the routine returns information about all the buttons in the top-level collection.

`reportData`

Raw HID report bytes.

Description

Returns a value capability array that describes all HID control values that meet a specified selection criteria.

---

getUsagesEx

```
TypeScript
getUsagesEx(reportType: HID.ReportType, linkCollection: number, reportData: Uint8Array):
IHIDUsageAndPage[];
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

`linkCollection`

Specifies the link collection of the button usages. If `linkCollection` is nonzero, the routine only returns information about the buttons that this link collection contains; otherwise, if `linkCollection` is zero, the routine returns information about all the buttons in the top-level collection.

`reportData`

Raw HID report bytes.

Description

Returns a list of the all the HID control button usages that are set to ON in a HID report.

---

getButtons

```
TypeScript
getButtons(reportType: HID.ReportType, usagePage: number, linkCollection: number, reportData:
Uint8Array): number[];
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

`usagePage`

Specifies the usage page of the button usages. The method only returns information about buttons on this usage page.

`linkCollection`

Specifies the link collection of the button usages. If `linkCollection` is nonzero, the routine only returns information about the buttons that this link collection contains; otherwise, if `linkCollection` is zero, the routine returns information about all the buttons in the top-level collection.

`reportData`

Raw HID report bytes.

Description

Returns a value capability array that describes all HID control values that meet a specified selection criteria.

---

getButtonsEx

```TypeScript
getButtonsEx(reportType: HID.ReportType, linkCollection: number, reportData: Uint8Array):
IHIDUsageAndPage[];
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

`linkCollection`

Specifies the link collection of the button usages. If `linkCollection` is nonzero, the routine only returns information about the buttons that this link collection contains; otherwise, if `linkCollection` is zero, the routine returns information about all the buttons in the top-level collection.

`reportData`

Raw HID report bytes.

Description

Returns a list of the all the HID control button usages that are set to ON in a HID report.

---

getUsageValue

```TypeScript
getUsageValue(reportType: HID.ReportType, usagePage: number, usage: number, linkCollection:
number): number;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

`usagePage`

Specifies the value's usage page.

`usage`

Specifies the usage of the value.

`linkCollection`

Specifies the link collection that contains the value. If `linkCollection` is nonzero, the routine only searches for the usage in this link collection; otherwise, if `linkCollection` is zero, the routine searches for the usage in the top-level collection. reportData: Uint8ArrayRaw HID report bytes.

Description

Extracts the data associated with a HID control value that matches the selection criteria in a HID report.

---

getScaledUsageValue

```
TypeScript
getScaledUsageValue(reportType: HID.ReportType,
    usagePage: number,
    usage: number,
    linkCollection: number,
    reportData: Uint8Array): number;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

`usagePage`

Specifies the usage page of the value to extract.

`usage`

Specifies the usage of the value to extract.

`linkCollection`

Specifies the link collection identifier of the value to extract. A `linkCollection` value of zero identifies the top-level collection.

`reportData`

Raw HID report bytes.

Description

Returns the signed and scaled result of a HID control value extracted from a HID report.

---

getUsageValueArray

```
TypeScript
getUsageValueArray(reportType: HID.ReportType,
    usagePage: number,
    usage: number,
    linkCollection: number,
    reportData: Uint8Array): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

`usagePage`

Specifies the usage page of the usage value array.

`usage`

Specifies the usage of the usage value array.

`linkCollection`

Specifies the link collection that contains the usage value array. If `linkCollection` is nonzero, the routine only searches for a usage value array in this link collection; otherwise, if `linkCollection` is zero, the routine searches for a usage value array in the top-level collection.

`reportData`

Raw HID report bytes.

Description

Extracts the data associated with a HID control usage value array from a HID report.

HID.ReportType

IHIDBuilder Interface

Description

This interface is implemented by HID Session object and allows you to construct HID reports to be sent to the device.

Declaration

TypeScript

```
interface IHIDBuilder {

    // Methods
    setData(reportType: HID.ReportType, data: IHIDData[]): Uint8Array;
    setUsageValue(reportType: HID.ReportType, usagePage: number, usage: number, linkCollection:
number, v: number): Uint8Array;
    setScaledUsageValue(reportType: HID.ReportType, usagePage: number, usage: number,
linkCollection: number, v: number): Uint8Array;
    setUsageValueArray(reportType: HID.ReportType, usagePage: number, usage: number,
linkCollection: number, data: Uint8Array): Uint8Array;
    setUsages(reportType: HID.ReportType, usagePage: number, linkCollection: number, data:
number[] | Uint16Array): Uint8Array;
    setButtons(reportType: HID.ReportType, usagePage: number, linkCollection: number, data:
number[] | Uint16Array): Uint8Array;
    unsetUsages(reportType: HID.ReportType, usagePage: number, linkCollection: number, data:
number[] | Uint16Array): Uint8Array;
    unsetButtons(reportType: HID.ReportType, usagePage: number, linkCollection: number, data:
number[] | Uint16Array): Uint8Array;
}
```

C#

```
// This interface is not available in managed environment
```

C++

```
// This interface is not available in native environment
```

IHIDBuilder Methods

---

setData

```TypeScript
setData(reportType: HID.ReportType, data: IHIDData[]): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`reportType`

The type of the report.

`data`

Array of data objects that specify which buttons and usage values to set.

Description

Sets a specified set of HID control button and value usages in a HID report.

---

setUsageValue

```TypeScript
setUsageValue(reportType: HID.ReportType, usagePage: number, usage: number, linkCollection: number, v: number): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

`usagePage`

Specifies the usage page of a usage.

`usage`

Specifies the usage.

`linkCollection`

Specifies the link collection that contains the usage. If `linkCollection` is nonzero, the routine only sets the usage, if one exists, in this link collection. If `linkCollection` is zero, the routine sets the first usage it finds in the top-level collection.

`v`

The value to set.

Description

Sets a HID control value in a specified HID report.

---

setScaledUsageValue

```TypeScript
setScaledUsageValue(reportType: HID.ReportType, usagePage: number, usage: number,
linkCollection: number, v: number): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

`usagePage`

Specifies the usage page of a usage.

`usage`

Specifies the usage.

`linkCollection`

Specifies the link collection that contains the usage. If `linkCollection` is nonzero, the routine only sets the usage, if one exists, in this link collection. If `linkCollection` is zero, the routine sets the first usage it finds in the top-level collection.

`v`

The value to set.

Description

Converts a signed and scaled physical number to a HID usage's logical value, and sets the usage value in a specified HID report.

---

setUsageValueArray

```TypeScript
setUsageValueArray(reportType: HID.ReportType, usagePage: number, usage: number,
linkCollection: number, data: Uint8Array): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

`usagePage`

Specifies the usage page of a usage.

`usage`

Specifies the usage.

`linkCollection`

Specifies the link collection that contains the usage. If `linkCollection` is nonzero, the routine only sets the usage, if one exists, in this link collection. If `linkCollection` is zero, the routine sets the first usage it finds in the top-level collection.

`data`

The value array to set.

Description

Sets a HID control usage value array in a specified HID report.

---

setUsages

```
TypeScript
setUsages(reportType: HID.ReportType, usagePage: number, linkCollection: number, data:
number[] | Uint16Array): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

`usagePage`

Specifies the usage page for the usages.

`linkCollection`

Specifies the link collection that contains the usages. If `linkCollection` is nonzero, the routine only sets the usages, if they exist, in this link collection. If `linkCollection` is zero, the routine sets the first usage for each specified usage in the top-level collection.

`data`

Data array to set.

Description

Sets specified HID control buttons ON (1) in a HID report.

setButtons

```TypeScript
setButtons(reportType: HID.ReportType, usagePage: number, linkCollection: number, data:
number[] | Uint16Array): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

`usagePage`

Specifies the usage page for the usages.

`linkCollection`

Specifies the link collection that contains the usages. If `linkCollection` is nonzero, the routine only sets the usages, if they exist, in this link collection. If `linkCollection` is zero, the routine sets the first usage for each specified usage in the top-level collection.

`data`

Data array to set.

Description

Sets specified HID control buttons ON (1) in a HID report.

unsetUsages

```TypeScript
unsetUsages(reportType: HID.ReportType, usagePage: number, linkCollection: number, data:
number[] | Uint16Array): Uint8Array;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

`usagePage`

Specifies the usage page for the usages.

`linkCollection`

Specifies the link collection that contains the usages. If `linkCollection` is nonzero, the routine

only sets the usages, if they exist, in this link collection. If `linkCollection` is zero, the routine sets the first usage for each specified usage in the top-level collection.

`data`

Data array to set.

Description

Sets specified HID control button usages OFF (zero) in a HID report.

---

unsetButtons

```
TypeScript
unsetButtons(reportType: HID.ReportType, usagePage: number, linkCollection: number, data:
number[] | Uint16Array): Uint8Array;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`reportType`

Type of the report.

`usagePage`

Specifies the usage page for the usages.

`linkCollection`

Specifies the link collection that contains the usages. If `linkCollection` is nonzero, the routine only sets the usages, if they exist, in this link collection. If `linkCollection` is zero, the routine sets the first usage for each specified usage in the top-level collection.

`data`

Data array to set.

Description

Sets specified HID control button usages OFF (zero) in a HID report.

HID.ReportType IHIDData

IHIDCaps Interface

Description

This interface is implemented by HID capability object. You can get HID capabilities using the IHIDDevice.caps property.

Declaration

```typescript
interface IHIDCaps {
    // Properties
    usage: number;
    usagePage: number;
    inputReportLength: number;
    outputReportLength: number;
    featureReportLength: number;
}
```

```
// This interface is not available in managed environment
```

```
// This interface is not available in native environment
```

IHIDCaps Properties

---

usage

```typescript
usage: number;
```

```
// This property is not available in managed environment
```

```
// This property is not available in native environment
```

Description

Specifies the top-level collection's usage page.

---

usagePage

```typescript
usagePage: number;
```

```
// This property is not available in managed environment
```

```
// This property is not available in native environment
```

Description

Specifies a top-level collection's usage ID.

---

inputReportLength

```
TypeScript
inputReportLength: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies a maximum length, in bytes, of the input report.

---

outputReportLength

```
TypeScript
outputReportLength: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies a maximum length, in bytes, of the output report.

---

featureReportLength

```
TypeScript
featureReportLength: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies a maximum length, in bytes, of the feature report.

IHIDRange Interface

Description

This interface is implemented by HID range object. It specifies the range of a parameter (by specifying minimum and maximum values).

Declaration

```typescript
interface IHIDRange {
    // Properties
    min: number;
    max: number;
}
```

```csharp
// This interface is not available in managed environment
```

```cpp
// This interface is not available in native environment
```

IHIDRange Properties

---

min

```typescript
min: number;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Specifies the minimum value of a parameter.

---

max

```typescript
max: number;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Specifies the maximum value of a parameter.

IHIDValue Interface

Description

This interface is implemented by HID value object.

Declaration

```typescript
interface IHIDValue {
    // Properties
    isRange: boolean;
    value: IHIDRange | number;
}
```

```csharp
// This interface is not available in managed environment
```

```cpp
// This interface is not available in native environment
```

IHIDValue Properties

---

isRange

```typescript
isRange: boolean;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

`true` if value property has the type IHIDRange and `false` if it is a number.

---

value

```typescript
value: IHIDRange | number;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Contains the actual parameter value. Can be either an instance of IHIDRange or a number, depending on the `isRange` property's value.

IHIDValueCaps Interface

Description

This interface is implemented by HID value capability object.

Declaration

TypeScript
```
interface IHIDValueCaps {
    // Properties
    usagePage: number;
    reportID: number;
    isAlias: boolean;
    bitField: number;
    linkCollection: number;
    linkUsage: number;
    linkUsagePage: number;
    isAbsolute: boolean;
    hasNull: boolean;
    bitSize: number;
    reportCount: number;
    unitsExp: number;
    units: number;
    logical: IHIDRange;
    physical: IHIDRange;
    usage: IHIDValue;
    string: IHIDValue;
    designator: IHIDValue;
    dataIndex: IHIDValue;
}
```

C#
```
// This interface is not available in managed environment
```

C++
```
// This interface is not available in native environment
```

IHIDValueCaps Properties

---

usagePage

TypeScript
```
usagePage: number;
```

C#
```
// This property is not available in managed environment
```

C++
```
// This property is not available in native environment
```

Description

Specifies the usage page of the usage or usage range.

---

reportID

TypeScript
```
reportID: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies the report ID of the HID report that contains the usage or usage range.

---

isAlias

```
TypeScript
isAlias: boolean;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Indicates, if true, that the usage is member of a set of aliased usages. Otherwise, if isAlias is false, the value has only one usage.

---

bitField

```
TypeScript
bitField: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Contains the data fields (one or two bytes) associated with an input, output, or feature main item.

---

linkCollection

```
TypeScript
linkCollection: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies the index of the link collection in a top-level collection's link collection array that contains the usage or usage range. If linkCollection is zero, the usage or usage range is contained in the top-level collection.

---

linkUsage

```
TypeScript
linkUsage: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies the usage of the link collection that contains the usage or usage range. If linkCollection is zero, linkUsage specifies the usage of the top-level collection.

---

linkUsagePage

```
TypeScript
linkUsagePage: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies the usage page of the link collection that contains the usage or usage range. If linkCollection is zero, linkUsagePage specifies the usage page of the top-level collection.

---

isAbsolute

```
TypeScript
isAbsolute: boolean;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies, if true, that the usage or usage range provides absolute data. Otherwise, if isAbsolute is false, the value is the change in state from the previous value.

---

hasNull

```
TypeScript
hasNull: boolean;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies, if true, that the usage supports a NULL value, which indicates that the data is not valid and should be ignored. Otherwise, if hasNull is false, the usage does not have a NULL value.

---

bitSize

```
TypeScript
bitSize: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies the size, in bits, of a usage's data field in a report. If reportCount is greater than one, each usage has a separate data field of this size.

---

reportCount

```
TypeScript
reportCount: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies the number of usages that this structure describes.

---

unitsExp

```TypeScript
unitsExp: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Specifies the usage's exponent, as described by the USB HID standard.

---

units

```TypeScript
units: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Specifies the usage's units, as described by the USB HID Standard.

---

logical

```TypeScript
logical: IHIDRange;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Specifies a usage's signed lower and upper bounds.

---

physical

```TypeScript
physical: IHIDRange;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies a usage's signed lower and upper bounds after scaling is applied to the logical range.

---

usage

```
TypeScript
usage: IHIDValue;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Usage.

---

string

```
TypeScript
string: IHIDValue;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

String descriptor.

---

designator

```
TypeScript
designator: IHIDValue;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Designator.

---

dataIndex

```
TypeScript
dataIndex: IHIDValue;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Data index.

IHIDRange

IHIDNode Interface

Description

This interface is implemented by HID node object.

Declaration

```
TypeScript
interface IHIDNode {
    // Properties
    linkUsage: number;
    linkUsagePage: number;
    parentIndex: number;
    numberOfChildren: number;
    nextSibling: number;
    firstChild: number;
    collectionType: number;
    isAlias: boolean;
}
```

```
C#
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

IHIDNode Properties

---

linkUsage

```
TypeScript
linkUsage: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies the usage ID of a top-level collection.

---

linkUsagePage

```
TypeScript
linkUsagePage: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies the usage page of the collection.

---

parentIndex

```
TypeScript
parentIndex: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies the index of the collection's parent collection. If the collection has no parent, Parent is zero.

---

numberOfChildren

```
TypeScript
numberOfChildren: number;
```

```
C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Specifies the number of child collections that the collection contains.

---

nextSibling

```TypeScript
nextSibling: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Specifies the index of the collection's immediate sibling. If the collection has no sibling, NextSibling is zero.

---

firstChild

```TypeScript
firstChild: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Specifies the index of the collection's first child collection. If the collection has no children, FirstChild is zero.

---

collectionType

```TypeScript
collectionType: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Specifies the type of collection item.

isAlias

```typescript
isAlias: boolean;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Specifies, if `true`, that this collection is an aliased collection. Otherwise, if `false`, the collection is not aliased.

IHIDData Interface

Description

This interface is implemented by HID data object.

Declaration

```typescript
interface IHIDData {
    // Properties
    index: number;
    data: number;
}
```

```csharp
// This interface is not available in managed environment
```

```cpp
// This interface is not available in native environment
```

IHIDData Properties

index

```typescript
index: number;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Specifies the data index of a control.

---

data

```
TypeScript
data: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies a data value.

IHIDButtonCaps Interface

Description

This interface is implemented by HID button capability object.

Declaration

```
TypeScript
interface IHIDButtonCaps {
    // Properties
    usagePage: number;
    reportID: number;
    isAlias: boolean;
    bitField: number;
    linkCollection: number;
    linkUsage: number;
    linkUsagePage: number;
    isAbsolute: boolean;
    usage: IHIDValue;
    string: IHIDValue;
    designator: IHIDValue;
    dataIndex: IHIDValue;
}
```

```
C#
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

IHIDButtonCaps Properties

---

usagePage

```typescript
usagePage: number;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Specifies the usage page for a usage or usage range.

---

reportID

```typescript
reportID: number;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Specifies the report ID of the HID report that contains the usage or usage range.

---

isAlias

```typescript
isAlias: boolean;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Indicates, if true, that a button has a set of aliased usages. Otherwise, if isAlias is false, the button has only one usage.

---

bitField

```typescript
bitField: number;
```

```csharp
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Contains the data fields (one or two bytes) associated with an input, output, or feature main item.

---

linkCollection

```
TypeScript
linkCollection: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies the index of the link collection in a top-level collection's link collection array that contains the usage or usage range. If linkCollection is zero, the usage or usage range is contained in the top-level collection.

---

linkUsage

```
TypeScript
linkUsage: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies the usage of the link collection that contains the usage or usage range. If linkCollection is zero, linkUsage specifies the usage of the top-level collection.

---

linkUsagePage

```
TypeScript
linkUsagePage: number;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies the usage page of the link collection that contains the usage or usage range. If linkCollection is zero, linkUsagePage specifies the usage page of the top-level collection.

---

isAbsolute

```
TypeScript
isAbsolute: boolean;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Specifies, if true, that the button usage or usage range provides absolute data. Otherwise, if isAbsolute is false, the button data is the change in state from the previous value.

---

usage

```
TypeScript
usage: IHIDValue;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Usage.

---

string

```
TypeScript
string: IHIDValue;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

String descriptor.

---

designator

```typescript
designator: IHIDValue;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Designator.

---

dataIndex

```typescript
dataIndex: IHIDValue;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Data index.

IHIDValue

IHIDUsageAndPage Interface

Description

This interface is implemented by HID usage result object.

Declaration

```typescript
interface IHIDUsageAndPage {
    // Properties
    usagePage: number;
    usage: number;
}
```

```csharp
// This interface is not available in managed environment
```

```cpp
// This interface is not available in native environment
```

IHIDUsageAndPage Properties

usagePage

```TypeScript
usagePage: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Usage page.

usage

```TypeScript
usage: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Usage value.

TypeScript

Starting from version 7.13, Device Monitoring Studio allows you to use TypeScript for user scripts. TypeScript is an open-source programming language, created by Microsoft which is based on new standard of EcmaScript and is a superset of JavaScript. Starting from version 7.70 TypeScript is always used.

It provides syntax sugar to simplify defining classes, interfaces, lambda functions and so on. Strong type checking is helpful for finding and fixing bugs before even running user scripts.

Version 7.25 upgrades the built-in TypeScript compiler to version 1.5.

Version 7.70 upgrades the built-in TypeScript compiler to version 1.8.10.

Syntax Check

Device Monitoring Studio internally has TypeScript declarations for all supported objects, their properties and methods. This allows automatic syntax and type check when user script is launched.

License

This product uses the open-source software TypeScript language from Microsoft. The product license is provided below and is also available on-line on project web-site.

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

   "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   You must give any other recipients of the Work or Derivative Works a copy of this License; and

   You must cause any modified files to carry prominent notices stating that You changed the files; and

   You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the

appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Monaco Editor

License

This product uses the open-source software Monaco Editor from Microsoft. The product license is provided below and is also available on-line on project web-site.

The MIT License (MIT)

Copyright (c) 2016 Microsoft Corporation

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Remote Monitoring

Device Monitoring Studio supports an infrastructure for remote monitoring of serial and USB devices[1]. A running instance of Device Monitoring Studio may establish a connection to Device Monitoring Studio Server installed on a remote server, obtain a list of server's serial and USB devices and start a monitoring session for any supported device.

Remote Source provides this support inside Device Monitoring Studio, while Device Monitoring Studio Server, installed on a remote computer, "shares" serial and USB devices, locally connected to that computer.

Network Setup

DMS establishes a TCP connection to a server. Server may be configured to listen on specific local endpoints (address and port), or on all local interfaces. Network administrator must ensure that incoming connections are allowed for the configured endpoints. You can find more information in the Server Configuration topic.

Server Deployment

Device Monitoring Studio Server is available as a stand-alone package. It includes serial and USB monitoring components (which may be installed individually), server executable (capable of running as stand-alone process or installed as Windows Service) and server configuration utility.

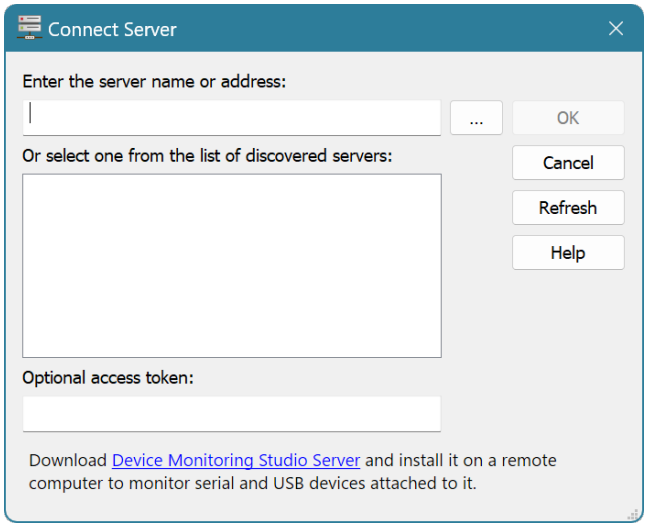
More information on server installation and supported deployment scenarios can be found in the corresponding topic.

---

1. This feature is not available in all product editions.↩

Connect Server Window

This window allows you to connect to a Device Monitoring Studio Server running on a remote computer.



Specify the address of a remote server. Use one of the following supported syntaxes:

`servername`

Name of the remote server.

`servername:port`

Name of the remote server and TCP port to use for connection.

`ipv4`

IPv4 address of a remote server.

`ipv4:port`

IPv4 address and TCP port of a remote server.

`ipv6`

IPv6 address of a remote server.

`[ipv6]:port`

IPv6 address and TCP port of a remote server.

You can also press the **...** browse button to select the computer.

Device Monitoring Studio also displays a list of servers it auto-discovers on a local network. If server's advertisement is enabled and network is configured properly, its address will appear in a list.

Provide an optional *access token* if the server you are connecting to is configured with token-based access control.

Device Monitoring Studio Server

Device Monitoring Studio Server is a stand-alone package that can be installed on a PC to share its local serial and USB devices over the network. Once installed and configured, server handles monitoring requests from remote instances of Device Monitoring Studio.

Downloading Device Monitoring Studio Server

You can download Device Monitoring Studio Server from HHD Software Ltd. web site:

Download Server

The server installation package consists of the following components:

Main server executable

Contains server code. May be launched as a stand-alone process or installed as Windows Service (default mode).

Server configuration utility

While server reads its configuration from a simple JSON file, this configuration utility provides a visual way to edit the configuration file.

Serial monitoring components

A number of components that enable monitoring of serial devices.

USB monitoring components

A number of components that enable monitoring of USB devices.

Network Configuration

By default, Device Monitoring Studio Server's installer adds an exclusion to the Windows Firewall during installation (this option may be switched off). In more complex network setups, network administrator must ensure incoming connections to the server executable are allowed.

Interoperability with Device Monitoring Studio

Device Monitoring Studio Server cannot be installed side-by-side with Device Monitoring Studio (and vice versa). However, starting with version 9.12, Device Monitoring Studio installer contains an optional

server component.

By default, server is not installed as Windows Service and instead may be run on demand.

Installation

Device Monitoring Studio Server must be installed on a computer in order to allow its local serial and USB devices to be monitored from remote instances of Device Monitoring Studio. An administrator access is required to install Device Monitoring Studio Server.

Device Monitoring Studio Server is available to download from HHD Software Ltd. web site:

Download Server

The server installation package consists of the following features:

Main server executable

Contains server code. May be launched as a stand-alone process or installed as Windows Service (default mode). This feature is required.

Server configuration utility

While server reads its configuration from a simple JSON file, this configuration utility provides a visual way to edit the configuration file. This feature is checked by default.

Serial monitoring components

This feature includes a number of components that enable monitoring of serial devices. This component is checked by default.

USB monitoring components

This feature includes a number of components that enable monitoring of USB devices. This feature is checked by default.

Install as Windows Service

If checked, setup installs server as Windows Service. The service is running every time the system starts (under Local System account), no matter if the user logs in or not. If not checked, the server may still be manually launched. This feature is checked by default.

Add Windows Firewall exception

If checked, a new “Allow” rule is added to Windows Firewall, allowing all incoming connections to server's listening endpoints. This feature is checked by default.

Disable anonymous access

If checked, server disables anonymous access. Use the server configuration file or Server Configuration Utility to explicitly set up server security. If unchecked, server allows anonymous connections. This feature is unchecked by default.

Server cannot be installed on the same computer where Device Monitoring Studio is already installed. If you need to share serial and USB devices on a computer with Device Monitoring Studio installed, use the optional server component, available in Device Monitoring Studio installer.

If server is installed as part of Device Monitoring Studio, it is not installed as Windows Service by default.

Server Security

Device Monitoring Studio Server supports two basic security settings: *anonymous access* and *token-based access*.

Anonymous Access

In this mode (which is the default), server accepts all incoming connections and allows monitoring of all its devices. This mode is only recommended in a fully controlled network environment, for example, in a local network in a lab.

Token-Based Access

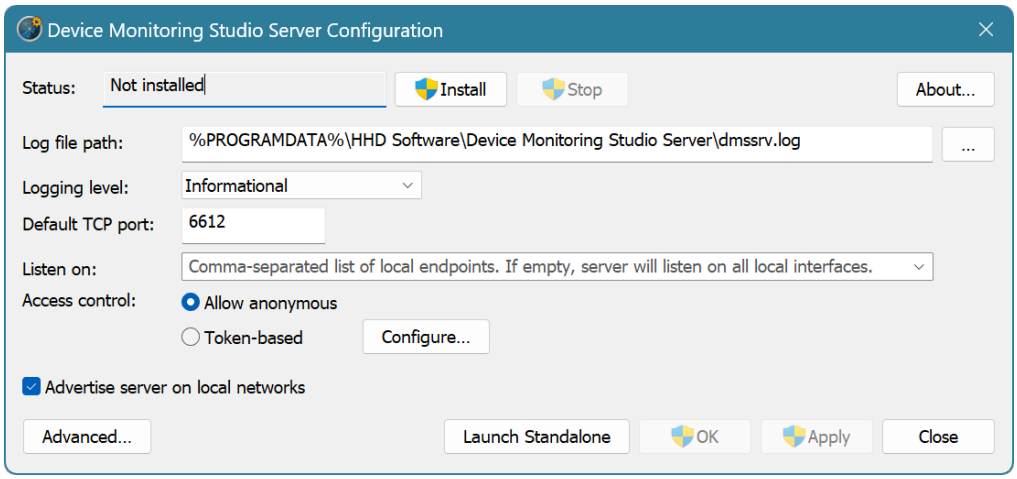
In this mode, server administrator needs to create a number of *access tokens* and optionally provide a list of devices each access token is allowed to monitor. Access token is any unique string which must be passed by a connecting client.

If client does not pass an access token or passed access token is not found on the server, it is refused connection. If the device the client wants to monitor is not explicitly allowed for his access token, the client receives an "access denied" error.

Server security is configured in server configuration file and may also be configured using Server Configuration Utility.

Configuration Utility

This stand-alone utility provides a visual way to configure Device Monitoring Studio Server. It is basically a visual editor for server configuration file (`config.json`).



Status

This field shows the current status of server if installed as Windows Service.

Install

Installs Device Monitoring Studio Server as Windows Service.

Remove

Uninstalls Device Monitoring Studio Server as Windows Service.

Start

Starts the service.

Stop

Stops the service.

Log file path

The full path to the log file. Supports environment strings (enclosed in `%`). Note that the server will fail to start if it cannot obtain write access to the file.

Logging level

Logging level. Higher level produces more messages and includes all messages from all lower levels.

The following logging levels are supported (from lowest to highest):

Level	Description
Critical	Only critical error messages are included
Error	All error messages are included
Warning	Warning messages are included
Informational	Informational messages are included
Debug	All messages are included

Default TCP port

The port to use if a given endpoint misses port specification.

Listen on

A comma-separated list of local endpoints the server listens on. The server supports the following endpoint syntaxes:

`hostname`

Any local hostname that the server will try to resolve. Must refer to one of the local network adapters.

`hostname:port`

The same as before, but explicitly specifies TCP port.

`ipv4`

A local IPv4 address.

`ipv4:port`

A local IPv4 address and TCP port.

`ipv6`

A local IPv6 address.

`[ipv6]:port`

A local IPv6 address and TCP port. Note that IPv6 address must be enclosed in square brackets if it is followed by a port number.

`*` **or empty string**

Use all local addresses. This is the default setting.

`*:port`

Use a specific TCP port on all local addresses.

Access control

Select whether server allows anonymous access or requires token-based access.

Configure...

Press to configure token-based access control.

Advertise server on local networks

If enabled, the server will advertise itself on local networks. Please note that server advertisement is subject to network configuration. If the client does not see a server with advertisement turned on, it still may successfully connect to it using direct address.

Advanced

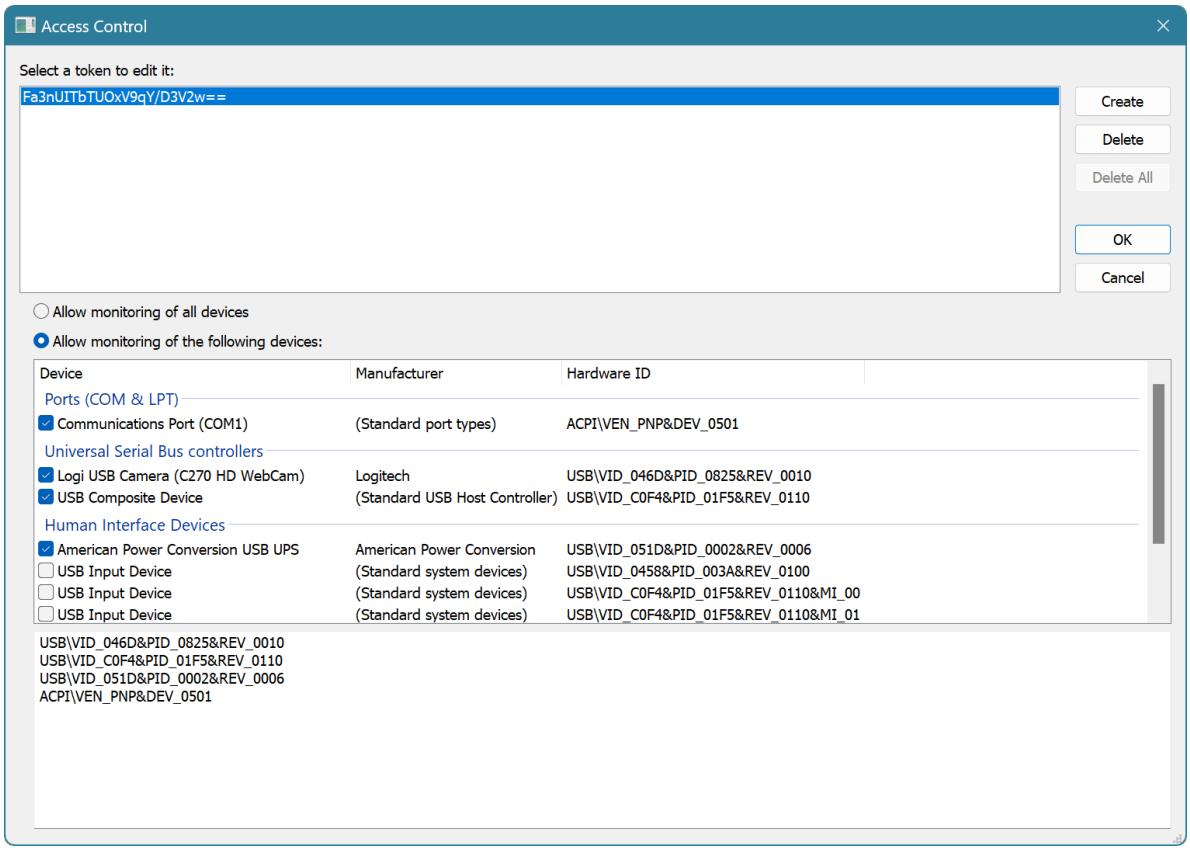
Launches external editor to directly edit `config.json` file.

Launch Standalone

If server is not currently installed as Windows Service and is not running, launches an instance of a server as a standalone process.

Token-Based Access Control Editor

This window is displayed when you press the **Configure...** button.



The top part of the window allows you to manage a list of access tokens. Click the **Create** button to create new access token, **Delete** button to delete selected token and **Delete All** button to delete all access tokens.

When token is selected, you can choose if it allows all devices to be monitored or only a subset of server devices. Internally, the allowed device list is configured using device *hardware IDs*. You can either check corresponding devices in a list or enter hardware IDs manually.

Server Configuration File Reference

Device Monitoring Studio Server reads its configuration from the `config.json` file located in the same folder as `dmssrv.exe` server executable (unless overridden by `--config` command-line parameter).

This topic describes the structure of the configuration file. Device Monitoring Studio Server Configuration Utility may also be used as a visual editor of the configuration file.

NOTE

JSON file format does not support comments. If you use `//` or `/* ... */` comments in a configuration file, server will treat it as malformed and will refuse to start.

WARNING

> Any configuration file syntax error is considered a critical error.

JSON File Structure

We will describe the JSON file structure by means of a TypeScript declaration:

```TypeScript
interface AccessToken
{
    // A list of hardware IDs this token is allowed to monitor.
    // An empty list means that all devices are allowed
    allowedHardwareIds?: string[];
}

const enum MessageLevel
{
    Critical,   // 0
    Error,  // 1
    Warning,    // 2
    Informational,  // 3
    Debug,  // 4
}

interface Config
{
    // Full path to log file. Default value is "%PROGRAMDATA%\HHD Software\Device Monitoring Studio Server\dmssrv.log"
    logPath?: string = "%PROGRAMDATA%\\HHD Software\\Device Monitoring Studio Server\\dmssrv.log";

    // Logging level, specified as number. Default value is 3 (Informational)
    logLevel?: MessageLevel = 3;

    // Comma-separated list of local endpoints, see below. Default value is an empty string
    listenEndpoints?: string = "";

    // default TCP port. Default value is 6612
    defaultListeningPort?: number = 6612;

    // A list of access tokens. Default value is an empty array
    authKeys?: AccessToken[] = [];

    // Enable server advertisement. Default value is true
    advertise?: boolean = true;

    // Enable anonymous access. Default value is true
    allowAnonymous?: boolean = true;
};
```

All configuration parameters are optional and provide reasonable defaults if omitted.

Endpoint Syntax

`listenEndpoints` parameter is a comma-separated list of local endpoints the server listens on. The server supports the following endpoint syntaxes:

`hostname`

Any local hostname that the server will try to resolve. Must refer to one of the local network adapters.

`hostname:port`

The same as before, but explicitly specifies TCP port.

`ipv4`

A local IPv4 address.

`ipv4:port`

A local IPv4 address and TCP port.

`ipv6`

A local IPv6 address.

`[ipv6]:port`

A local IPv6 address and TCP port. Note that IPv6 address must be enclosed in square brackets if it is followed by a port number.

`*` or empty string

Use all local addresses. This is the default setting.

`*:port`

Use a specific TCP port on all local addresses.

Server Configuration

The server reads configuration file only on startup. If you modify the configuration file, make sure the server is restarted to pick up the changes.

Server Command-Line Reference

The `dmssrv.exe` server executable supports the following command-line parameters:

Parameter	Value	Description
`-?, --help`		Displays the list of supported parameters with short description.
`--nologo`		Do not display the logo message.
`-config`, `--config`	`path`	Full path to the JSON config file. If omitted, config.json from the server startup folder is used.
Logging		
`--log-path`	`path`	Write server log to the specified file.
`--log-level`	`LOGGING-LEVEL`	Set logging level to one of the following: **critical** only critical errors **error** all errors **warnings** errors and warnings **info** informational messages | **debug** maximum information for debugging
`--no-screen-log`		Do not display a copy of log to the console.
Service operations		
`-install-service`, `--install-service`		Install server as Windows Service.
`-uninstall-service`, `--uninstall-service`		Uninstall service.

User Interface

Notification Windows

Notification Windows are used to pay user's attention to any activity in the Device Monitoring Studio. Each notification window displays important information about the current state of the application.

You can control the time each notification window is displayed in Tools » Settings, General Tab - see *Display notification window for N seconds* option.

You can also control if you want to display specific notification window in **Tools » Settings, General Tab** or directly in the displayed notification window.

Available Notifications

Click on the item below to see the description of specific notification window:

- Next Connected Device (Serial)
- Next Connected Device (USB)
- Line View Notification
- New Terminal Session
- Continue playback
- Statistics Special Mode
- Statistics Static Line
- Fast data entering (MODBUS Send window)

Next Connected Device (Serial)

This notification window briefly lists actions you need to perform to work with the "Next Connected Device" session type. Plug in the PnP-compliant serial device you want to monitor to start receiving data for the monitoring session.

Next Connected Device (USB)

This notification window briefly lists actions you need to perform to work with the "Next Connected Device" session type. Plug in the USB device you want to monitor to continue the monitoring session.

Line View Notification

Line View visualizer is different from other visualizers in that it does not open a visualizer window. Instead, it adds the line state signals to the application scrollbar. If you have multiple running monitoring sessions, it displays the line states for the currently active session.

This notification informs you about the location of the Line View visualizer you added to the monitoring session.

New Terminal Session

To create new serial terminal session, click the button the notification points to, or use the **Tools » Serial Terminal » New Terminal Session** command.

Continue Playback

This notification window shows you the location of the **Next** button, which you can use to continue paused log file playback. This button can also be used to skip the long delay in the monitored data.

Statistics Special Mode

This notification is displayed when you start the Statistics visualizer and tells you that Special Mode is available for high-speed monitoring (Only the Statistics visualizer must be configured for the session to activate this mode).

Statistics Static Line

This notification window is displayed when you click on the plot area of the Statistics visualizer and tells you that you can also click while holding a **Ctrl** key to place another kind of value tracking line.

Fast data entering (MODBUS Send window)

This notification tells you that the *fast data entering mode* is currently active.

Commands

All functions of the Device Monitoring Studio are invoked through commands. For convenience, almost every command is accessible using several user interface tools:

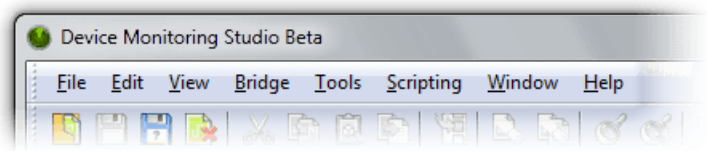
- Each command is accessible through the main menu. Several commands used to execute local actions of specific tool windows are accessible through context, or shortcut menu. Shortcut menu is invoked by clicking the right mouse button, or by pressing "Context Menu" keyboard button (usually located next to the right **Ctrl** key).
- Most commands are also displayed on toolbars. Two kinds of toolbars are used in the Device Monitoring Studio. Main toolbar is located right under the menu bar and is fully customizable. You can also create new toolbars and place them near the menu bar. Other toolbars, which are not customizable, are located in tool windows and contain the most used commands that are specific for those tool windows.
- Several most used commands have associated keyboard shortcuts or hot keys. Pressing such associated key or key combination invokes the command. You can change assigned key combinations as well as assign new key combinations to other commands.

All commands described in this documentation are mentioned by their location in the main menu. For example, the "Open..." command, located under the File main menu item is referenced as **File » Open...**. Other examples are **File » Exit** for an "Exit" command under File menu and **Tools » Settings...** for a "Settings..." command under Tools menu.

By convention, if ellipsis symbol (...) terminates menu item's name, it means that user is required to provide additional information to execute this command. This may include displaying a dialog box, asking for confirmation etc. For example, the **File » Exit** command does not require any additional action from you, while the **File » Open...** command opens a File Open dialog, therefore requiring you to provide additional information.

Menus

Main menu bar is located at the top of application window, right under the title bar.



Main menu contains almost all commands implemented by the Device Monitoring Studio. It is structurally divided into the following groups:

File

Contains commands related to working with workspaces.

Edit

Contains all editing commands and Clipboard commands.

View

Contains the list of tool windows, commands to export/import tool window configuration or load a predefined configuration.

Bridge

Contains commands related to Serial Bridge.

Tools

Contains generic commands that control different tool windows, as well as a current monitoring session. The **Tools » Settings...** command is an entrance to the central place where you fine-tune the Device Monitoring Studio by changing different options.

Scripting

Contains scripting-related commands.

Window

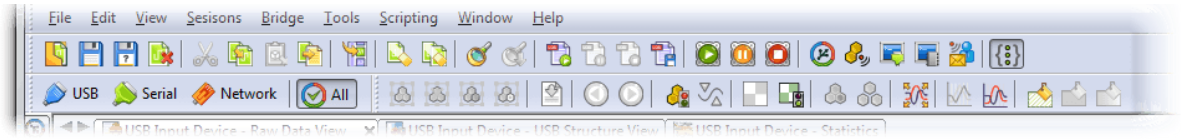
Contains commands that manage the number and location of editor windows.

Help

Contains commands that can be used to access this documentation file, check for program updates, display keyboard map (a table of associated shortcuts) as well as display information about the Device Monitoring Studio.

Toolbars

Toolbars present a subset of Device Monitoring Studio's commands to the user by means of displaying commands' images. Main toolbar and user created toolbars are located under the menu bar:



To execute a command located on the toolbar left-click its image. If you hold a mouse pointer over an image for a while, command's name is displayed in a tooltip window, along with assigned keyboard shortcuts. More detailed command description is displayed on the status bar at the same time.

Main toolbar, as well as user-defined toolbars, are fully customizable. You can quickly change the order of commands on the toolbar using the following procedure:

1. Press and hold an Alt key.
2. Locate the mouse cursor over the command you want to move to another location.
3. Press the left mouse button.
4. Drag mouse to another location and see how the command moves within the toolbar.
5. Release a left mouse button to “drop” a command on its new location. Release an Alt key as well. If you moved mouse away from the toolbar, the command is removed from the toolbar.

You can also create toolbar separators using this procedure.

Advanced toolbar customization is described in the Toolbar Customization topic. For example, in addition to command arrangement you can smoothly change the size of toolbar images.

Toolbars located in tool windows are not customizable and contain commands related to the tool window where they are located.

Keyboard Shortcuts

Device Monitoring Studio associates a number of its commands with keyboard shortcuts. You can always see the list of all currently assigned shortcuts using the Help » Keyboard Map command. The list of assigned shortcuts may also be printed for reference or copied to the Clipboard to be used in another application.

Keyboard Customization section illustrates how you can change default keyboard shortcuts or create your own.

Tool Windows

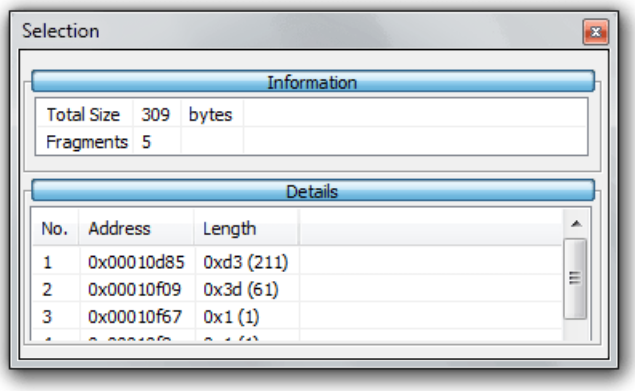
Tool windows are a special-purpose windows that are always visible to the user (unless they are hidden) and always ready to perform a task. The following Device Monitoring Studio features and functions are implemented as tool windows:

- Devices Tool Window
- Sessions Tool Window
- USB Device Descriptor
- USB Configuration Descriptor
- USB HID Descriptor
- USB Dependent Devices
- Serial Device Information
- Packet Builder
- MODBUS Send

Location

A tool window either may be *floating* or *docked* to the application frame. A docked tool window may also be *auto-hidden*. All these terms are described later in this section.

Floating Tool Window



Floating tool window is convenient when you want to maximize the working space used by data visualizers. It may be positioned outside of the main application window, and may even be located on another monitor if you have one. This is contrary to the *docked tool window*, which may only be located within application window.

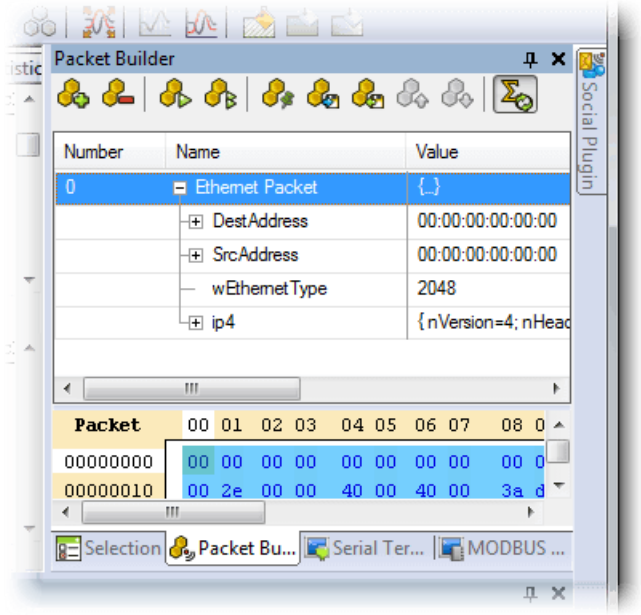
The position of the floating tool window may be changed by dragging it by its title with a mouse. You can also use mouse to change the size of the floating tool window by dragging either its frame or one of four corners.

The important thing to have in mind while working with floating tool windows is the *keyboard focus*. Operating system directs all keyboard input to the window owning keyboard focus. Usually an *active title*

is used to indicate the window that has keyboard focus. Blinking caret may also be used to indicate the focused window. Activating floating tool window by a mouse click always brings a keyboard focus to this window, allowing you to provide input to the window. Clicking outside of the floating tool window usually takes a keyboard focus away from it. To continue sending keyboard input to the tool window, you may need to activate it again.

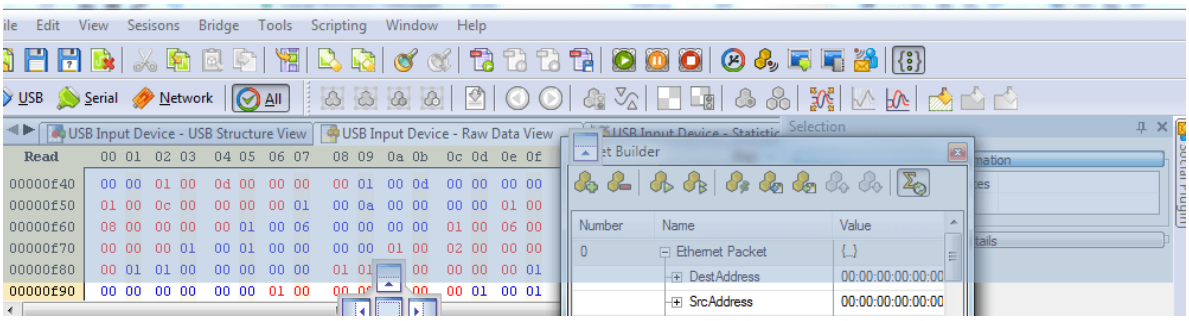
Docked Tool Window

A tool window may also be docked. When docked, it is attached to a *tool window frame*. A frame may be attached next to another frame or directly to the top, left, right or bottom side of application window. Frames may be located next to each other, inserted one into another and merged. Each frame contains at least one docked tool window and may contain several. Below is a screenshot of a frame with a single tool window docked to the right side of application window:

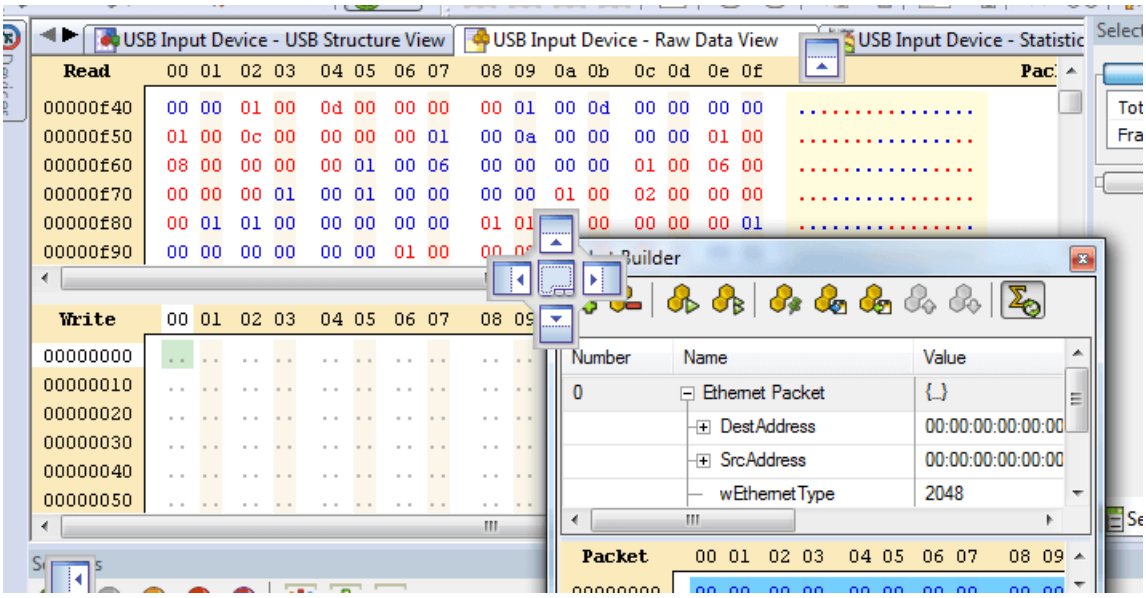


You can dock any tool window by dragging it by its title bar or tab. During dragging, special placeholders appear on the screen. When you move the mouse pointer over the placeholder, semi-transparent rectangle appears on the screen, indicating an approximate position of the docked window. Releasing the mouse button docks a window in the indicated position.

Side placeholders always appear, allowing you to dock a tool window into a new frame and attach it to the top, left, right or bottom side of the application window.



When you drag a window over a docked frame or over the workspace area, another placeholder appears. It is composed of five boxes: top, left, middle, right and bottom. Dragging a window over the side box docks the window into a new frame, which is attached to the top, left, right or bottom of the frame currently under mouse pointer.



If you drop a window to the middle box, you will dock it into the frame currently under mouse pointer. Dropping a window to the docked window's title also docks it to target window's frame.

You can use mouse to change position of docked windows within a docked frame, as well as drag the window “out of the frame” to make it floating or to dock it into another frame or new frame.

Double-clicking on the docked frame's title makes a frame floating. Double-clicking on the docked window's tab makes only that window floating. Double-clicking on the floating frame's title docks it back to the same location it previously occupied.

All docked frames are separated with each other by splitters. When you position a mouse pointer over such splitter, it changes its shape, telling you that you can drag the splitter to change size of adjoining frames. Splitter position is proportional to the application window size; frames are resized automatically and proportionally when you resize application window.

Auto-Hidden Tool Windows

Any docked tool window frame may be auto-hidden. To auto hide a frame, click the Auto Hide button:

Four auto-hide bars are located on four application window sides. They are hidden when empty and appear as soon as you add at least one frame to it. When you auto-hide a docked frame, it chooses one of four auto-hide bars, depending on the docked position. That is, if the frame was docked closer to the right side of application window, right auto-hide bar is chosen and so on.

Each docked window's tab in the auto-hidden frame is displayed on the auto-hide bar. Two auto-hidden

frames are separated from each other with a slightly larger gap. Dragging mouse over the tab opens corresponding tool window. The tool window smoothly drives out of the auto-hide bar. Click in the tool window to switch keyboard focus to it if you need to provide any keyboard input to the window. The window is automatically closed (quickly drives back into the auto-hide bar) when keyboard focus is lost by the window and mouse cursor is not over it.

Auto hiding tool windows allow you to save the screen space, while still having quick access to functionality provided by the tool windows.

Pressing Auto Hide button again "unhides" the docked frame and returns it to its original docked position.

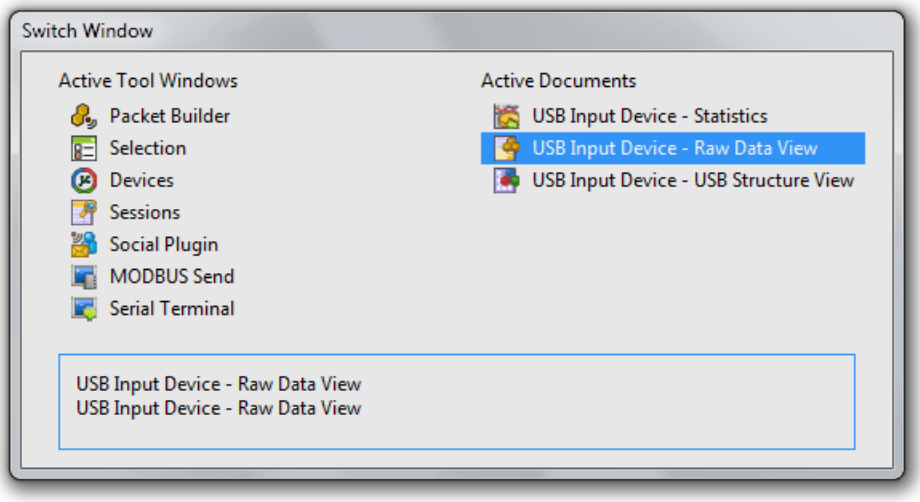
Tool Window Visibility

Any tool window is always in one of two states: visible or hidden. Visible means the tool window is floating or docked to any of the docked frames or auto-hidden and appears on one of four auto-hide bars. Hidden tool window does not appear anywhere on the screen. Note that a tool window, which is not directly visible by the user, but which tab is visible in one of the docked frame, considered visible by this definition.

Pressing the Close button hides the tool window or entire frame, depending on the "Close affects the current tool window only" setting on the General settings page. To make a previously hidden window visible, select its name in the View menu. This restores the previous position, docked and auto-hidden state of the window and activates it. It also activates the tool window even if it is in visible state, therefore, allows you to quickly move keyboard focus to the tool window, without using mouse.

Window Switching

Device Monitoring Studio interface presents you with a large number of tool windows, as well as an unlimited number of data visualizer windows. Navigation Window - a convenient window switching mechanism is provided for you.



Navigation window is opened when you press the **Ctrl + Tab** or **Ctrl + Shift + Tab** key combination (see also Keyboard Customization). It lists all visible (possibly auto-hidden) tool windows as well as all opened editor windows. Using the arrow keys, **Tab**, **Shift + Tab** or mouse, you may select the window you want to activate. As soon as you release a **Ctrl** key, the navigation window is closed and selected window becomes active.

Navigation window provides a quick window activation mechanism.

Workspace

A workspace is full application configuration that includes the following:

1. A state and configuration of all running monitoring sessions. This configuration includes monitoring devices, data visualizers, exporters and window locations.
2. Current tool window configuration.
3. Serial Terminal sessions.
4. Scripting configuration.
5. Packet Builder configuration.

You may control the inclusion of some of these components into workspaces on the **Tools » Settings, General Tab**.

Working with Workspaces

A current application configuration may be saved to a workspace file using the **File » Save Workspace** or **File » Save Workspace As...** commands.

An existing workspace may be loaded using the **File » Open...** command.

Global Switch

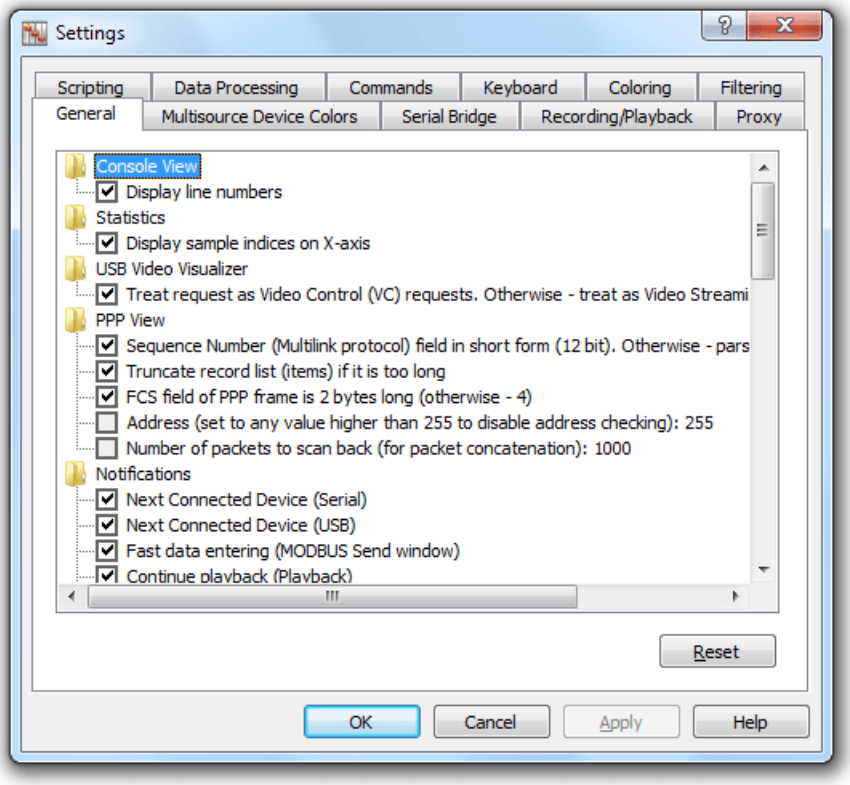
Device Monitoring Studio consists of three main components: Network, Serial and USB. While three separate installers are available for download, which install only corresponding components, a single "all-in-one" installer is also available.

If you have installed the complete Device Monitoring Studio, but usually (or always) work with a single component, use the **Global Switch** to hide unneeded components:

Pressing the **Network**, **Serial** or **USB** buttons leaves only devices and windows that relates to selected data type. Alternatively, click the **All** button to revert to default "all-in-one" configuration.

Configuration

General Tab



This configuration page allows you to set up general application properties. All properties are organized in the tree and can be either Boolean (on/off), selectable (one of) or numeric.

General Group

Display splash screen on startup

If enabled, a splash screen is displayed when Device Monitoring Studio is launched.

Foreground downloading

Device Monitoring Studio takes all available bandwidth when it downloads a new version.

Define tooltip interval

A number of millisecond a tooltip stays opened

Save tools configuration with workspace

If enabled, Tool window configuration (visibility and location) is saved into the workspace.

Last recently used workspaces to display

A number of last recently used workspaces to keep and display in the **File** menu.

Display notification window for (seconds)

A number of seconds a notification window is visible.

Close affects the current tool window only

If enabled and when close button is pressed on a stacked tool window frame, only the current tool window is closed, otherwise, the entire frame is closed.

Restore main window position on startup

If enabled, Device Monitoring Studio's main window position is automatically restored to its last position.

Ask to save modified workspace

If enabled and the current workspace has been modified, a user is asked to save the changes before closing Device Monitoring Studio.

Check for updates

Select how often Device Monitoring Studio checks whether a new version is available.

Join consequent requests

Automatically join consequent packets of the same type.

Notifications Group

This group contains the flags controlling the appearance of different notification windows.

Statistics Group

This group contains options related to Statistics visualizer.

Display sample indices on X-axis

Check this option to display the sample ordinal number on X-axis. Packet number and packet time are always displayed.

Raw Data View

Display lowercase hexadecimal

If enabled, all hexadecimal numbers use lowercase letters, otherwise they use uppercase.

Display popup packet information

Display detailed packet information when mouse is hovered on top of packet data.

Network Packet View

Display popup packet information

Display detailed packet information when mouse is hovered on top of packet data.

MODBUS

These options are used to control the legacy MODBUS View data visualizer.

Truncate register/coil/request list if it is too long

Automatically limit the number of items displayed.

Add base offsets for registers, discrete inputs, etc.

TBD

Parse requests on WRITE (responses on READ) direction

If enabled, MODBUS View data visualizer treats the current monitoring session as MODBUS Master, otherwise, it treats it as MODBUS Slave.

Concatenate packets

Automatically join split MODBUS packets.

RTU mode

If enabled, treat the current monitoring session as MODBUS RTU mode, otherwise treat it as

MODBUS ASCII mode.

Parse as hex

Display all numbers in hexadecimal format.

Console View

This section contains options for Legacy Console View data visualizer.

Display line numbers

If enabled, line numbers are displayed in the data visualizer.

Scripting

Reload scripts on startup

If enabled, Device MMonitoring Studio automatically opens all previously opened script files.

Ask to save unsaved scripts on close

If enabled and there are unsaved scripts, a prompt is displayed to save the changes.

Include scripting configuration into workspace

If enabled, all opened scripts are saved into the workspace.

Serial Terminal

Include terminal configuration into workspace

If enabled, all running [terminal sessions] are saved into the workspace.

Send text lines from file one by one

If enabled, a text file is split into lines and they are subsequently sent to the serial session one by one. Otherwise, the entire text file is sent at once.

Convert tab characters to N spaces

If not equals to zero, tab characters are replaced with a given number of space characters when `TAB` key is pressed in Serial Terminal Window.

Data block size

A size of the buffer (in bytes) to use in serial terminal session.

USB Audio Visualizer

Treat requests as Audio Control (AC) requests

If enabled, requests are treated like Audio Control (AC) requests. Otherwise, they are treated like Audio Stream (AS) requests.

USB Video Visualizer

Treat requests as Video Control (VC) requests

If enabled, requests are treated like Video Control (VC) requests. Otherwise, they are treated like Video Stream (VS) requests.

PPP View

Sequence number (Multilink protocol) field in short form (12 bit)

Parse sequence number (Multilink protocol) field in short form (12 bit) if option is enabled, otherwise parse it in long form (24 bit).

Truncate record list if it is too long

Limit the number of records in the list.

FCS field of PPP frame is 2 bytes long

If enabled, FCS field of PPP frame is 2 bytes long, otherwise it is 4 bytes long.

Address

Set to the device address to automatically do address checking or any number higher than 255 to disable it.

Number of packets to scan back

A number of packets PPP View looks back in search for a starting packet.

Auto-Hide

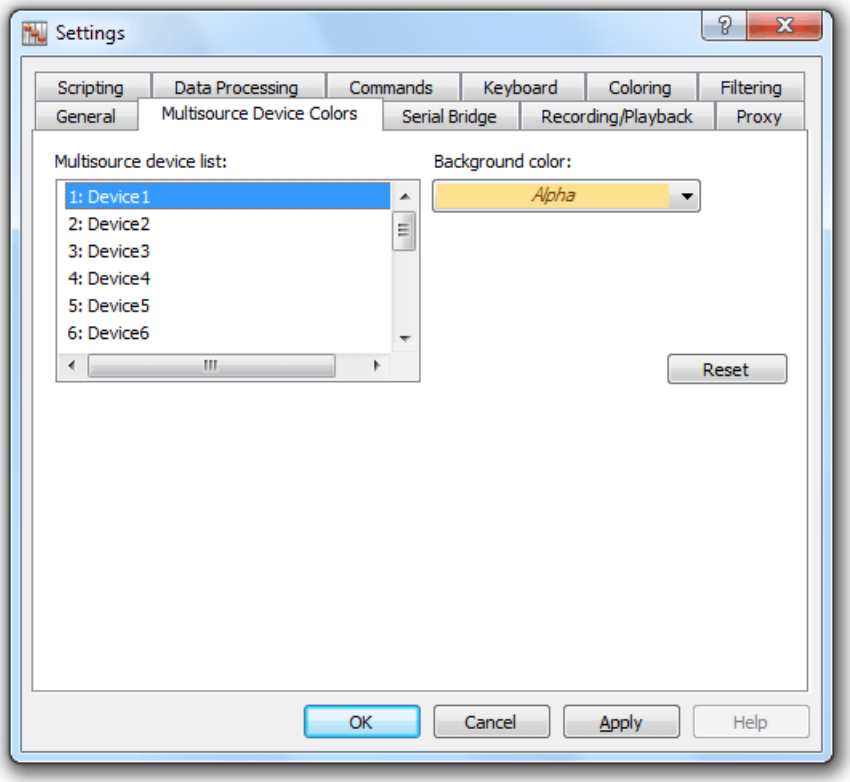
Hover time (ms)

A number of milliseconds a mouse needs to be hovered over auto-hidden tool window to open it.

Close time (ms)

A number of milliseconds a tool window stays visible until it closes after the mouse leaves.

Multi-Source Device Colors Tab



Use this configuration window to set up device colors for multi-source monitoring session. Different colors help you distinguish a packet captured from one device from another.

Select a device on the left and use the **Background color** control to select the color. Use the **Reset** button to restore the default configuration.

Recording/Playback Tab

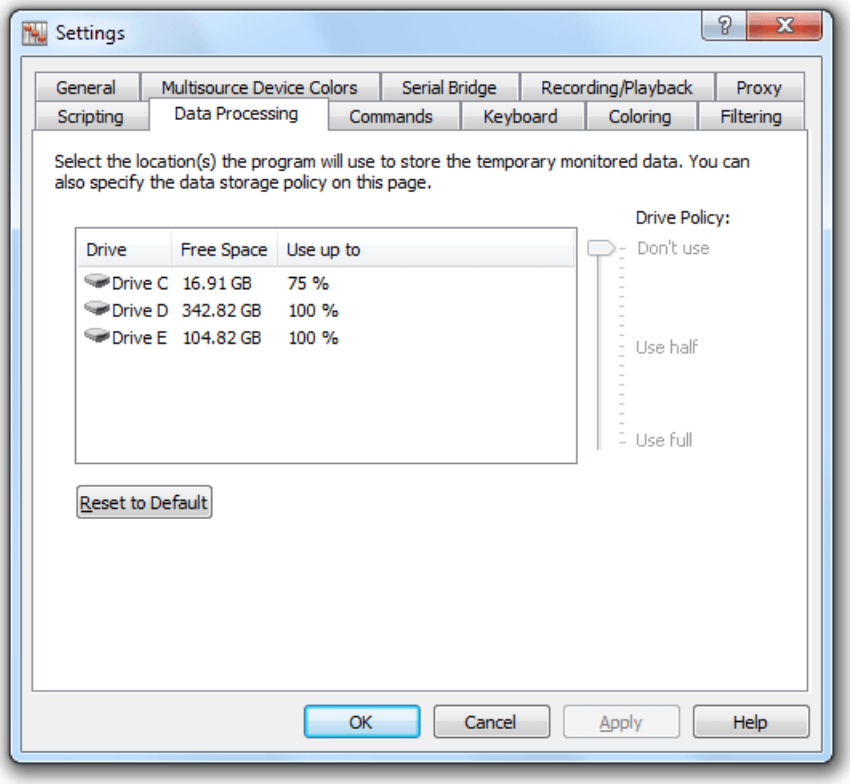
This page allows you to specify options for the Data Recording and Playback modules. Here you enter

the full path to the folder used to store the recorded log files and set the compression option.

Note that setting the compression option will slow down the recording. Do not use Compression when recording high-speed communications. Compression is NTFS compression and will work only on NTFS-formatted volumes.

This window also allows you to specify separate folder where Playback data source will look for log files to be shown in Devices Tool Window. It is recommended to have these paths to point to the same folder.

Data Processing Tab



The application uses the disk space to store temporary monitored data. It can be configured to place its temporary files on any available volume and configured to take as much free space as you want. Note that if the application becomes short of temporary storage, it starts deleting the oldest data, resulting in the removed data is not available for visualizers anymore.

Configuring Device Monitoring Studio Data Processing Policy

There can be set two major policies for data processing. See the information below to properly choose the data policy you need:

1. *Display as much as you can*. Set each available volume to allow as much space as you can. In this mode, the entire monitoring session will always be available for any visualizer you configure for your monitoring session. You will always be able to scroll to any location and see the packets monitored at any time.
2. *Consume as less space as possible*. Set all but one volumes to “Don't use” and one volume to a small value. This will make the Device Monitoring Studio to reuse the available space as soon as possible. Use this scenario if you are interested in recent monitored data only.

If you change data policy settings, it will immediately affect any new monitoring sessions you create.

When you close the monitoring session, all used disk space is freed.

Temporary Storage

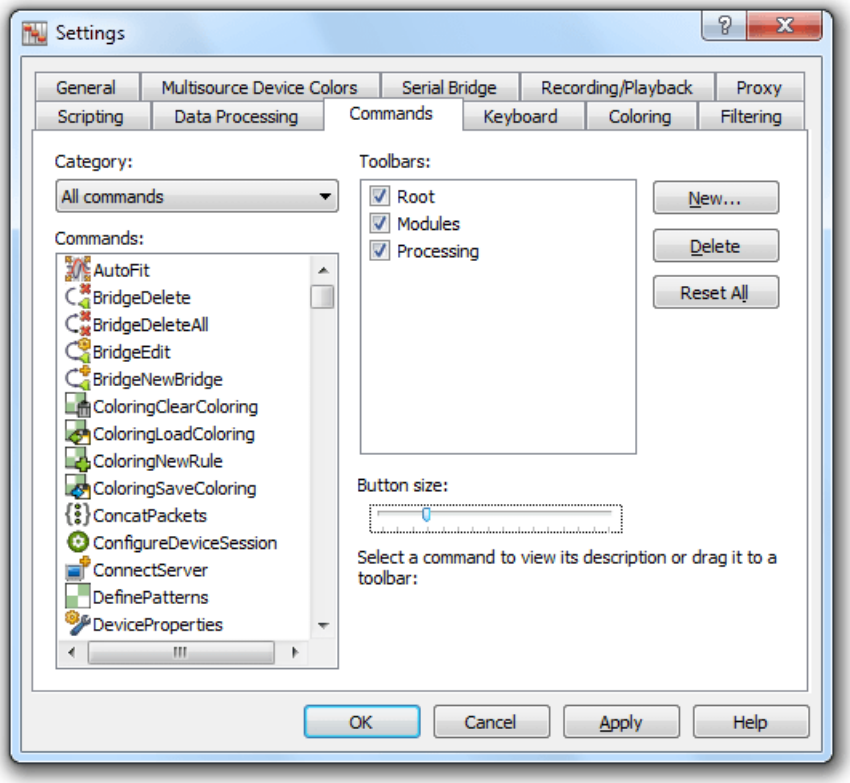
The system creates a hidden folder `$DMSTEMP$` on each configured volume where it creates its temporary files. When the monitoring session is closed, all files are automatically deleted. The hidden folder may remain until the application is closed. In case of unexpected shutdown, it is always safe to remove the folder manually.

In order to increase throughput, Device Monitoring Studio interleaves its access to different volumes. If multiple volumes refer to the same physical disk, you may increase throughput by disabling all but one volumes of that disk.

High-Performance Mode

When monitoring session runs with data processing modules that do not require history (like Statistics, Data Recording, Raw Exporter or Text Exporter), a special *high-performance mode* is enabled. In this case, Device Monitoring Studio will not use temporary storage to store session's data.

Commands Tab



This page lets you configure the toolbars. You see the list of available commands (that can be filtered by specifying Category) and the list of toolbars.

Creating New Toolbar

To create a toolbar, press the **New...** button. Enter the toolbar's name. The created toolbar is empty. Add new commands to it.

Deleting Toolbar

To delete a toolbar, select it in the list and click the **Delete** button.

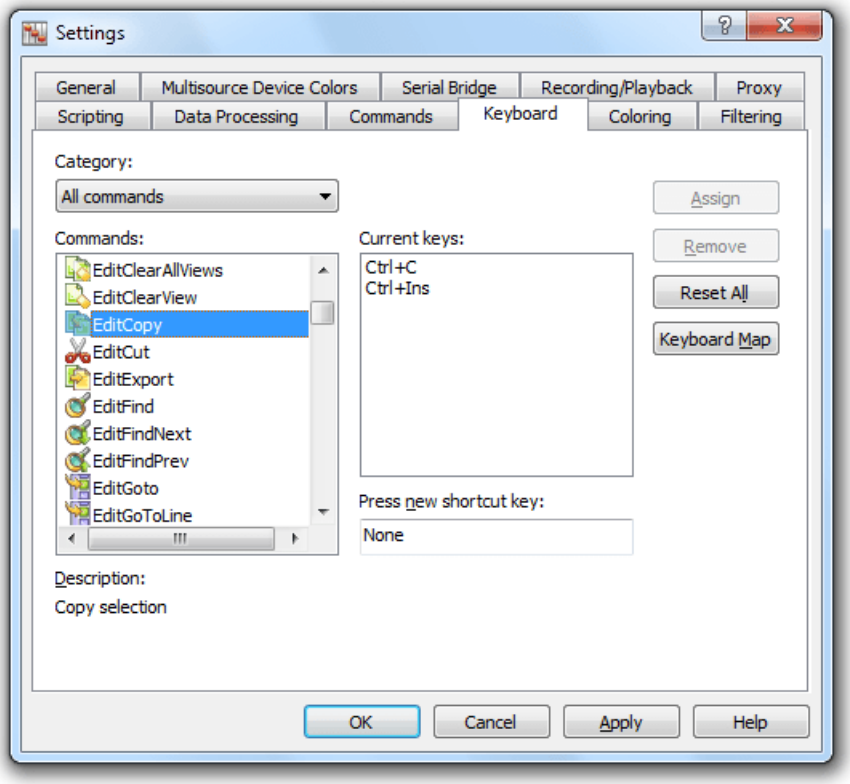
Configuring a Toolbar

- To remove a button from the toolbar, click on it and drag away from the toolbar.
- To move a button to another location, click and drag it.
- To add a button to a toolbar, click on the command in the Commands list and drag it to the toolbar.

Other Options

Hide a toolbar by unchecking the box next to its name in the Toolbars list. Change the size of the toolbar icons with the **Button Size** control. The Device Monitoring Studio's unique vector icon technology renders command icons at any size without artifacts.

Keyboard Tab

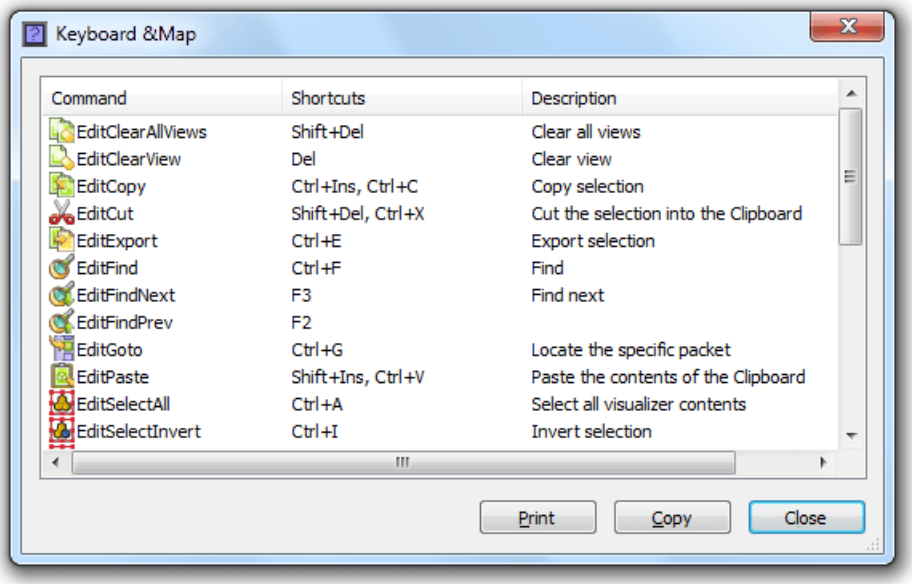


This page lets you configure the keyboard shortcuts. You see the list of available commands (that can be filtered by specifying Category) and the list of assigned key combinations.

Select the command in a list to view its current combinations. Delete existing combinations by selecting them in a list and pressing the **Delete** button. Type new combinations and assign them to the selected command.

Press the **Keyboard Map** button to bring up the keyboard map (also available through **Help » Keyboard Map** command of currently assigned shortcuts.

Keyboard Map

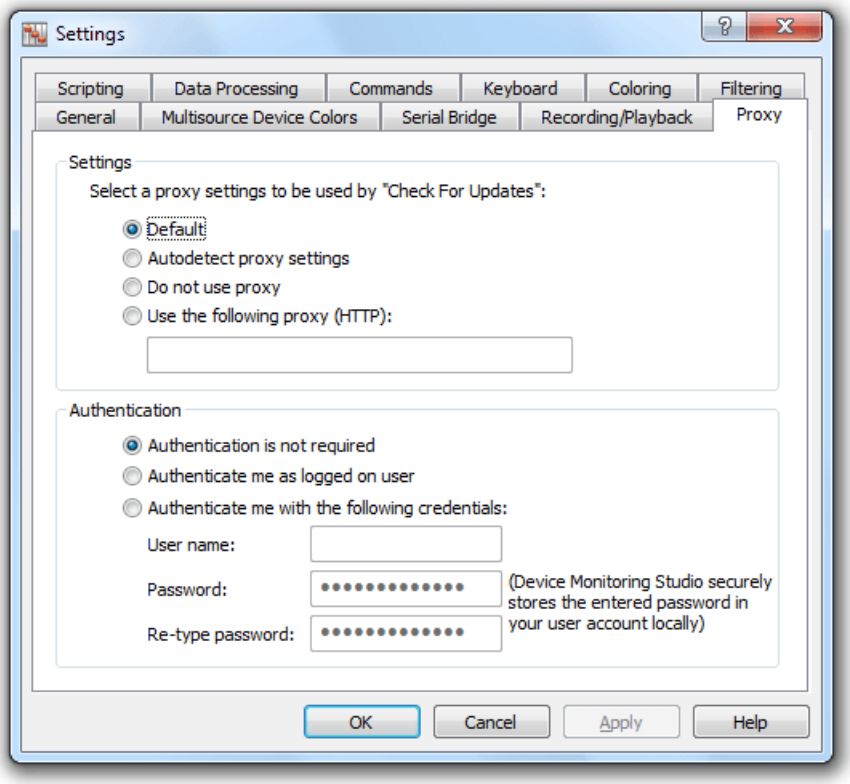


This window displays current association between the Device Monitoring Studio commands and keyboard shortcuts. You may select a subset of the items in a list and copy them to the **Clipboard**, or **Print**. If there is no selection, the entire window is copied or printed.

Proxy Tab

This page allows you to configure the proxy server used by features **Check for Updates** and **Downloader**. In most cases, you don't need to make any changes, as defaults will work on majority of installations.

When Device Monitoring Studio establishes a connection to remote server, it always does it over HTTP protocol over the 80 port. If your computer must use HTTP proxy when accessing HTTP resources, you may configure the proxy settings on this page. In addition, if your proxy server requires authentication, you may also configure credentials to use.



Server Settings

There are four options you can use to configure a proxy server:

Default

Application will use the proxy server configured in your default browser. This is a default setting.

Auto-detect proxy settings

Application will try to automatically detect proxy server settings.

Do not use proxy

Device Monitoring Studio will bypass any configured proxy.

Use the following proxy (HTTP):

Specify the address (and optionally port number) of the proxy server to use. Examples:

```
myserver.com
```

will use the HTTP proxy server `myserver.com` on port `80` (the default).

```
myserver.com:8080
```

will use the HTTP proxy server `myserver.com` on port `8080`

Proxy Server Authentication

There are three authentication options you can use:

Authentication is not required

Proxy Server does not require authentication.

Authenticate me as logged on user

Device Monitoring Studio will use the currently logged-on user to authenticate on the proxy server.

Authenticate me with the following credentials:

Enter the user name and password. You must enter the same password in both Password and Re-type password boxes. See the Security Considerations section below.

Security Considerations

When you configure the Device Monitoring Studio to use the entered credentials, it stores entered password in registry under the `HKEY_CURRENT_USER` key. It uses encryption to store the password. Only the same user is able to decrypt the password.

That is, if encrypted password is copied to another computer, an attacker will not be able to get the plain text password.