Table of Contents

Virtual Serial Port Tools Documentation

Virtual Serial Port Tools (VSPT) is the ultimate virtual serial ports creation and management application. It is capable of creating software-only virtual serial port devices and then using them to form advanced device configurations. It also allows sharing of local serial ports (physical, PnP or virtual) over the network over TCP/IP.

It builds upon a versatile high-performance virtual serial port driver, created by HHD Software Ltd. The driver operates exclusively in user-mode, raising the overall OS stability. The functionality of the driver may also directly be used by user code, allowing creation of custom virtual serial ports solutions. This is as simple as writing a small portion of TypeScript or JavaScript code that utilizes a simple yet very powerful Device Script API.

Below is a list of essential features provided by the toolkit:

Virtual Bridges

VSPT supports creation of either local or remote virtual bridges.

A virtual bridge is a pair of virtual serial port devices interconnected by means of a virtual null-modem cable. Virtual Serial Port Tools supports full emulation of serial port baud rate, line control parameters, flow control, buffer overflow and even allows setting specific level of noise emulation.

> **NOTE**
>
> Bridge always connects two **virtual** ports, which either reside on a single computer (local bridges) or two distinct computers (remote bridges).
>
> To make an advanced configuration where a virtual port is connected to a real port, see one of the next sections.

Use Create Local Bridge Window to create new local virtual serial bridge and Create Remote Bridge Window to create new remote virtual bridge.

The following API methods can be used to configure local serial bridges:

- ISerialPortLibrary.createBridgePort
- IBridgePortDevice.bridgePort

The following API methods and properties can be used to configure remote virtual serial bridges:

- ISerialPortLibrary.createBridgePort
- IBridgePortDevice.bridgeServer
- IBridgePortDevice.remoteLogin
- IBridgePortDevice.remoteDomain
- IBridgePortDevice.remotePassword
- IBridgePortDevice.isLocal

Alias Ports

A virtual serial port may be created as an alias to another existing serial port.

For example, you can create a virtual `COM2` port which will be an alias to an existing `COM1` port. Any application that successfully works with `COM1` can be switched to work with `COM2` without noticing any differences.

Use Create Alias Port Window to create new alias serial port.

The following API methods can be used to create and configure alias ports:

- ISerialPortLibrary.createAliasPort
- IAliasPortDevice.aliasPort

Shared Ports

It is well known that Windows operating system treats serial ports as **exclusive** devices, that is, it only allows a single application to "open" a port and communicate with a device.

However, this "exclusiveness" is sometimes a stop factor that prevents a very convenient setup. For example, you have a GPS Modem that constantly reports its location data to the serial port and you want two applications, *Application A* and *Application B* to receive this data.

Since it is impossible to remove the exclusive state of an existing COM port, Virtual Serial Port Tools solves this task by allowing you to create a new virtual serial port which **shares** any existing serial port to any number of applications.

For example, say our GPS Modem is connected to `COM1`. An invalid setup would be:

Application	Port	Result
Application A	`COM1`	*Success*
Application B	`COM1`	*Fail*

Now we create new virtual serial port `COM2` and set it to share `COM1` and the valid setup would be:

Application	Port	Result
Application A	`COM2`	*Success*
Application B	`COM2`	*Success*

Created virtual serial port "backs" any given existing serial device. Any application opening the shared port continues to work exactly like it worked with original device. At the same time, there is no longer a limit of a number of times a port may be opened by different or same applications.

Use Share Port Window to create and configure new shared virtual serial port.

The following API methods can be used to create and configure shared ports:

- ISerialPortLibrary.createSharedPort
- ISharedPortDevice.sharedPort

Split Ports

This type of serial port is similar to shared port described above, but also allows you to assign multiple device names to a shared serial port. In fact, when you create a split port configuration, the following happens under the hood:

First, a shared virtual serial device is created that provides shared access to a given serial device. Then a number of alias ports are created for the shared serial device. Any number of applications (or the same application multiple times) may open those alias ports, effectively accessing the same original serial device.

Use Split Port Window to create and configure new split virtual serial port.

The following API methods can be used to create and configure shared ports:

- ISerialPortLibrary.createSharedPort
- ISerialPortLibrary.createAliasPort

- ISharedPortDevice.sharedPort
- IAliasPortDevice.aliasPort

TCP/IP Ports (RFC2217, RAW)

VSPT allows the user to create a virtual serial port whose traffic is redirected to a specified TCP endpoint (host name/address and TCP port). The exchange protocol corresponds to RFC2217. A raw protocol is also supported.

The following two modes are supported for created TCP/IP virtual serial ports:

Connecting (TCP/IP client)

The user provides a remote host name or address and TCP/IP port to connect to. When application opens a virtual serial port, a TCP/IP connection is attempted to the specified endpoint. After successful connection, communication continues according to selected protocol.

Listening (TCP/IP server)

The user provides a local address (or `*` for all local addresses) and local TCP/IP port. When application opens a virtual serial port, a listening socket is created and driver waits for incoming TCP/IP connection. After successful connection, communication continues according to selected protocol.

Virtual Serial Port Tools also contains a TCP/IP Serial Ports Server component, which can be used to share local serial ports over TCP/IP network. This component may also be separately installed or even "copy-deployed" on a remote computer for simple port sharing scenarios.

Use Create TCP/IP Serial Port Window to create and configure new tcp serial port.

The following API methods can be used to create and configure TCP/IP ports:

- ISerialPortLibrary.createTcpPort
- ITcpPortDevice.remoteHost
- ITcpPortDevice.remoteTcpPort
- ITcpPortDevice.localAddress
- ITcpPortDevice.localTcpPort
- ITcpPortDevice.protocol
- ITcpPortDevice.reconnectTimeout

Script Ports

"Script ports" is the exclusive and powerful feature, only offered by the Virtual Serial Port Tools. Virtual script port is a port that is "powered" by a custom device script, written in TypeScript or JavaScript programming language.

A device script implements a virtual serial device, uncovering a tremendously wide list of capabilities and fast turn-around times for any custom serial port scenario, including creation of all kinds of device emulators, local and network communication channels and a lot, lot more.

Simple yet powerful and fully asynchronous Device Script API allows the device script to access the local file system, establish network connections and send HTTP requests.

Remote Ports

VSPT allows you to directly connect to any serial port that physically exists on another computer. This is done by means of creating a new local virtual serial port and directing it to a port shared by Remote Serial Ports Server component. This component may also be separately installed or even "copy-deployed" on a remote computer for simple port sharing scenarios.

Use Connect Remote Port Window to create and configure new remote virtual serial port.

The following API methods can be used to create and configure shared ports:

- ISerialPortLibrary.createRemotePort
- IRemotePortDevice.remoteHost
- IRemotePortDevice.remotePort
- IRemotePortDevice.login
- IRemotePortDevice.password
- IRemotePortDevice.domain

Pipe-connected Port

A new virtual serial port may be connected to a named pipe. This configuration mainly exists to simplify connection with virtual ports created by VM software in running guests.

Use Create New Pipe Port Window to create new pipe-connected port.

The following API functions can be used to create pipe-connected ports:

- ISerialPortLibrary.createPipePort
- IPipePortDevice.configureCreatePipe
- IPipePortDevice.configureCreatePipe2
- IPipePortDevice.configureConnectPipe

Listening Port

Finally, the last supported type of virtual serial port forms a “server”, listening part of a remote bridge.

A listening port first needs to be created on one computer and then remote bridge is created on another computer, connecting to this listening port.

Use Create Listening Port Window to create new listening port.

The following API functions can be used to create listening ports:

- ISerialPortLibrary.createBridgePort
- IBridgePortDevice.isListening
- IBridgePortDevice.securityDescriptor
- IBridgePortDevice.startListening

Supported Configurations

Virtual Bridges

Local Bridge

Virtual Serial Port Tools allows connecting two virtual serial devices to each other, forming a virtual bridge.

Connection

To create a virtual bridge or virtual null modem link, first create two bridge devices by calling ISerialPortLibrary.createBridgePort method twice and then connect them into the bridge by settings each device's IBridgePortDevice.bridgePort property to reference each other.

Operation

When an application writes data to the first serial port, all written data is transferred to the second port and vice versa. Bridge provides flow control and baud rate emulation as well as supports wait masks.

Configuration Utility

Configuration Utility allows the user to create local bridges, both non-permanent and permanent, using the Create Local Bridge Window.

Custom Pin-Out

Virtual Serial Port Tools supports custom pin-out configuration for local and remote bridges. See the corresponding topic for more details.

Remote Bridge

Two virtual serial ports, created on two computers may be joined together, forming a virtual remote bridge. Users may control who can access the created listening port (server-side) and supply credentials for a client-side port.

Connection

On a server side, a listening port must be created by calling a ISerialPortLibrary.createBridgePort method and then calling IBridgePortDevice.startListening method on a returned object.

On a client side, a new bridge port device must be created using ISerialPortLibrary.createBridgePort method and then configured by setting the following properties:

- IBridgePortDevice.bridgeServer - to set the name or address of the remote host
- IBridgePortDevice.bridgePort - to set the number of a listening port on a remote host
- IBridgePortDevice.remoteLogin - optional user name or login to authenticate on a remote host
- IBridgePortDevice.remoteDomain - optional domain name to authenticate on a remote host
- IBridgePortDevice.remotePassword - optional password to authenticate on a remote host

Operation

When an application writes data to the first serial port, all written data is transferred to the second port and vice versa. Bridge provides flow control and baud rate emulation as well as supports wait masks.

Configuration Utility

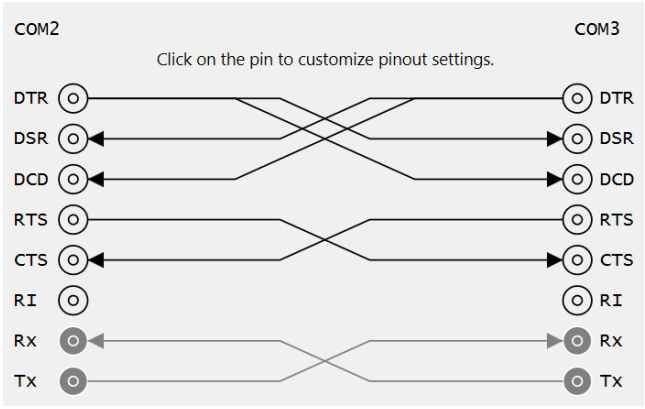
Configuration Utility allows the user to create remote bridges, both non-permanent and permanent, using the Create Remote Bridge Window and Create Listening Port Window.

Custom Pin-Out

Virtual Serial Port Tools supports custom pin-out configuration for local and remote bridges. See the corresponding topic for more details.

Bridge Pin-Outs

Virtual bridges created by Virtual Serial Port Tools (both local and remote) support custom "wiring" inside a virtual "cable". You may specify which pins are sending data to which other pins, both on local or remote socket. Below is a default configuration used:



Configuring Pin-Outs

If you are using the Configuration Utility, use Create Local Bridge Window or Create Remote Bridge Window to configure custom wiring.

Local and remote bridge pin-out is configured by settings values of the following properties: IBridgePortDevice.DTR, IBridgePortDevice.DSR, IBridgePortDevice.DCD, IBridgePortDevice.RTS, IBridgePortDevice.CTS and IBridgePortDevice.RI.

Validation

Virtual Serial Port Tools automatically validates the applied configuration and if it is invalid, you get an error from API.

Below is a table that shows you valid configuration for each pin:

Source Pin	Allowed Destinations
DTR (Data Terminal Ready)	DSR, DCD, CTS, RI
DSR (Data Set Ready)	DTR, RTS
DCD (Data Carrier Detect)	DTR, RTS
RTS (Request To Send)	DSR, DCD, CTS, RI
CTS (Clear To Send)	DTR, RTS
RI (Ring)	RI
RxD (Received Data)	None
TxD (Transmitted Data)	RxD (only remote)

In all cases (except TxD) if connection is allowed, it can be established both to the local or remote socket, effectively allowing the user to create loops.

Alias Ports

A virtual serial port may be created as an alias of another existing serial port.

Connection

To create an alias or mapped port, first call the ISerialPortLibrary.createAliasPort method and then specify the port parameters using the IAliasPortDevice.aliasPort property.

Additional port options can be set using IConfigurableDevice.baudRate, IConfigurableDevice.dataBits, IConfigurableDevice.parity, IConfigurableDevice.stopBits and IConfigurableDevice.flowControl properties. These properties allow you to override port parameters.

Operation

Created virtual serial ports acts as an alias to existing serial port for the calling application, providing full emulation of all serial port features supported by original device.

If there are parameter overrides configured for a port, they will silently be used instead of the configuration specified by the application.

A single port may have as many aliases as you need.

Configuration Utility

Configuration Utility allows the user to create and configure alias ports using the Create Alias Port Window.

Shared Ports

Virtual Serial Port Tools supports creation of virtual serial devices that “share” an existing serial device, allowing any number of applications to work with the original serial device.

Connection

To create a shared port, first call the ISerialPortLibrary.createSharedPort method and then configure it using the ISharedPortDevice.sharedPort property.

Additional port options can be set using IConfigurableDevice.baudRate, IConfigurableDevice.dataBits,

IConfigurableDevice.parity, IConfigurableDevice.stopBits and IConfigurableDevice.flowControl properties. These properties allow you to override port parameters.

Operation

Shared virtual serial port is created as non-exclusive device and allows itself to be opened by any number of applications. Actual data and control codes are redirected to original serial device.

The first application that opens a port is assigned as "master" application. Only master application is allowed to set several important port parameters, such as baud rate, line parameters and flow control settings. All attempts to modify these settings by other opening applications are silently discarded.

Other port parameters, like wait masks, timeouts and others are per-application and may be different. When all handles to the shared device are closed, the next application that opens a serial device is again assigned as "master" application.

If there are parameter overrides configured for a port, they will silently be used instead of the configuration specified by the application.

It is not supported to create several shared ports for the same original port. Configuration utility explicitly prohibits it, but API does not. If you still create several shared port devices for the same original device, only one of them will work at a time.

Configuration Utility

Configuration Utility allows the user to create and configure shared ports using the Share Port Window.

Split Ports

Split ports are not directly supported by Virtual Serial Port Tools. Instead, the configuration of a split port is achieved by means of first creating a shared port and then creating a number of aliases for the shared port.

Connection

First, create a shared port using the ISerialPortLibrary.createSharedPort and configure it using the ISharedPortDevice.sharedPort property.

Then create a required number of port aliases using ISerialPortLibrary.createAliasPort and point them to created shared port using the IAliasPortDevice.aliasPort property.

Operation

The split ports configuration operates exactly like the shared ports configuration. It can be used, for example, if an original device needs to be opened multiple times by a single application. Although a single shared port may be opened multiple times by a single application, applications are often designed in such a way to prevent the same port from appearing in user interface twice.

Split port configuration may be used in this scenario.

If there are parameter overrides configured for a port, they will silently be used instead of the configuration specified by the application.

Configuration Utility

Configuration Utility allows the user to create and configure split port configurations using the Split Port Window.

TCP/IP Ports

Virtual Serial Port Tools allows you to create virtual serial port device that redirects all traffic to a specified TCP endpoint, according to RFC2217 protocol (Telnet COM Port Control Option) as well as raw protocol. It also supports creating virtual serial devices that create a listening TCP/IP socket, bind it to a local address and local TCP/IP port and listen for incoming network connections.

HHD Software Ltd. also provides an implementation of TCP/IP Serial Ports Server, which is an optional component that can be installed on a remote computer to share its serial ports over the TCP/IP network.

Connection

To create a TCP port, first call the ISerialPortLibrary.createTcpPort method and then specify the port parameters.

- To create a connecting port, specify remote host/address and TCP port using the ITcpPortDevice.remoteHost and ITcpPortDevice.remoteTcpPort properties. Non-default reconnection timeout value may be set using the ITcpPortDevice.reconnectTimeout property.
- To create a listening port, specify local address and local port using the ITcpPortDevice.localAddress and ITcpPortDevice.localTcpPort properties.

Set the virtual serial port protocol using the ITcpPortDevice.protocol property. Currently, RFC2217 and raw protocols are supported.

Additional port options can be set using IConfigurableDevice.baudRate, IConfigurableDevice.dataBits, IConfigurableDevice.parity, IConfigurableDevice.stopBits and IConfigurableDevice.flowControl properties. These properties allow you to override port parameters.

Operation

Created virtual serial ports acts like the standard serial port for the calling application, providing full emulation of all serial port features supported by original device. Data protocol corresponds to RFC2217 or RAW, depending on the value of the ITcpPortDevice.protocol property.

For connecting ports, actual network connection is attempted at the time application opens the virtual serial port. If connection to remote server fails for any reason, the error code is propagated to the calling application.

For listening ports, a listening socket is created as soon as application opens the virtual serial port. As soon as remote party connects, the serial port communication is started.

Configuration Utility

Configuration Utility allows the user to create TCP ports and specify remote TCP endpoint using the Create TCP/IP Serial Port Window.

Remote Ports

Virtual Serial Port Tools allows you to directly connect to any serial port that physically exists on another computer. This is done by means of creating a new local virtual serial port and directing it to a port shared by Remote Serial Ports Server application running on a remote computer.

Connection

To create a remote port, first call the ISerialPortLibrary.createRemotePort method and then specify the port parameters using the IRemotePortDevice.remoteHost, IRemotePortDevice.remotePort, IRemotePortDevice.login, IRemotePortDevice.password, IRemotePortDevice.domain properties as well as optional IRemotePortDevice.connectionTimeout and IRemotePortDevice.connectionAttempts properties.

Operation

Created virtual serial ports acts like the original remote port for the calling application, providing full emulation of all serial port features supported by original device.

User credentials are required at the time of port creation. They are stored securely and used each time the port is opened by the application. A system administrator of the remote computer may assign which users are granted access to shared ports.

Actual connection is made at the time application opens the virtual serial port. If connection to remote server fails for any reason, the error code is propagated to the calling application.

Configuration Utility

Configuration Utility allows the user to create remote ports and specify remote endpoint and user credentials using the Connect Remote Port Window.

Script Ports

Virtual Script Port is the unique and powerful capability offered by Virtual Serial Port Tools. It allows the user to create a virtual serial port, “backed up” by a custom script, written in TypeScript or JavaScript.

Overview

A script that implements a virtual serial device logic (called device script further in this document) must be written in TypeScript or JavaScript (ES6) and must define a class that implements theIScriptDevice interface and define a createDevice or createDeviceAsync global function. This global function is called each time a virtual script port is opened by an application. This function must create an instance of the port class and return a reference to it.

Device creation function is allowed to perform some initialization and is passed an optional *initialization value* string. If it needs to perform an asynchronous initialization, use `createDeviceAsync` function, otherwise, use `createDevice` function.

VSPT APIISerialPortLibrary.createScriptPort method is used to create a virtual serial script port. Set an optional script initialization parameter using the IScriptPortDevice.initializationValue. Change the default script execution logging folder by setting the IScriptPortDevice.logPath property.

Once you have your port script ready, set it by calling IScriptPortDevice.setScriptFile or IScriptPortDevice.setScriptText method.

> **NOTE**
>
> The library compiles and validates the script and stores it internally. If `setScriptFile` method is successful, the original script file will never be read again.
>
> This is done for security reasons: by storing the script internally, VSPT protects the created virtual script port from unwanted script updates.
>
> If you make changes to a script file and want those changes applied, call `setScriptFile` method again.

If this method returns `false`, observe the IScriptPortDevice.validationErrors property to get a list of validation errors. This method may also throw an exception which usually means that while there were no syntax errors in a script, it still failed additional validation, like missing `createDevice` or `createDeviceAsync` global functions.

Script Structure

Below is the minimal device script:

```
///<reference path="hhdscriptport.d.ts" />

class MyDevice implements Port.IScriptDevice {   // 1
    // The following method is optional, see below
    public async onSend(data: Uint8Array): Promise<void> {  // 2
        // ...
    }

    // The following method is optional, see below
    public setParam(name: string, value: string): void {    // 3
        // ...
    }
}

// Either createDevice or createDeviceAsync global functions must be defined. If both are
defined, createDeviceAsync will be used
async function createDeviceAsync(initializationValue?: string): Promise<Port.IScriptDevice> { //
4
    return new MyDevice;
}

// Either createDevice or createDeviceAsync global functions must be defined. If both are
defined, createDeviceAsync will be used
function createDevice(initializationValue?: string): Port.IScriptDevice {    // 5
    return new MyDevice;
}
```

The following list describes the numbered elements in the sample above:

1. Script logic must be incorporated into a class which implements a Port.IScriptDevice interface.
2. An optional IScriptDevice.onSend method is used in the push model.
3. An optional IScriptDevice.setParam method, if present, allows the external code to send arbitrary data to the script, either by calling IScriptPortDevice.setScriptParam method or by sending a custom IOCTL_SCRIPTPORT_SET_PARAM device I/O request to the opened port.
4. `createDeviceAsync` global function, if present, will be called each time the virtual script port is opened. It is passed a copy of the IScriptPortDevice.initializationValue. This function must create an instance of a device class and return it.

   At least one of `createDeviceAsync` or `createDevice` global functions must be defined, otherwise, a validation error occurs. If both functions are defined, only `createDeviceAsync` will be used.
5. `createDevice` global function, if present, will be called each time the virtual script port is opened. It is passed a copy of the IScriptPortDevice.initializationValue. This function must create an instance of a device class and return it.

   At least one of `createDeviceAsync` or `createDevice` global functions must be defined, otherwise, a validation error occurs. If both functions are defined, only `createDeviceAsync` will be used.

> **NOTE**
>
> This script *execution context* is created after the port is opened and is destroyed as soon as it is closed. Any global variables you create will only live this long.

That is, you cannot store any state between port opening attempts. However, the execution context is guaranteed to exist until the port is closed by the application. Anyway, prefer storing any state in the created device script object, not on a global scope.

Send Data Model

Device script can use one of the following data flow models:

Push Model

In this model Virtual Script Port proactively sends all the data sent to the port to device script's `onSend` method.

Pull Model

In this model, device script queries (pulls) for the sent data.

The virtual serial port driver chooses the model by determining the presence of the IScriptDevice.onSend method in the device script. The sections below provide detailed explanation on flow models:

Push Model

Whenever an application sends any data to the virtual serial port, device script's IScriptDevice.onSend method is called with a copy of data sent. `onSend` method returns a promise. Once this promise is resolved, virtual serial port immediately discards sent data and completes the original application's Write request.

`Loopback Serial Port (Push Model).ts` sample script, included in the default product installation, illustrates the usage of a push model:

```
///<reference path="hhdscriptport.d.ts" />
/**
 * This script implements a simple "loopback" virtual serial device
 * It immediately returns everything that has been sent to it
 *
 * This version illustrates the usage of so-called "push" model
 * In this mode, virtual serial port driver calls an onSend method with data sent by application
to a serial port
 */
class LoopbackSerialDevicePush implements Port.IScriptDevice {
    public async onSend(data: Uint8Array): Promise<void> {
        // This script emulates an echo device, we simply provide the same data back
        port.provideReceivedData(data);
    }
}

function createDevice(): Port.IScriptDevice {
    return new LoopbackSerialDevicePush;
}
```

Pull Model

Whenever an application sends any data to the virtual serial port, it is put into the internal *output queue*. Application's Write request is completed immediately. At any time, a device script may query for the whole or part of the *output queue* by calling the IPort.getSentData method. This method returns a promise that is resolved with a copy of the *output queue*, or a part of it.

`Loopback Serial Port (Pull Model).ts` sample script, included in the default product installation, illustrates the usage of a pull model:

```
///<reference path="hhdscriptport.d.ts" />
/**
 * This script implements a simple "loopback" virtual serial device
 * It immediately returns everything that has been sent to it
 *
 * This version illustrates the usage of so-called "pull" model
 * In this mode, a port implementation calls the virtual serial port driver's getSentData
function that returns a Promise.
 * This promise is resolved with a byte array sent by application to a serial port
 */
class LoopbackSerialDevicePull implements Port.IScriptDevice {
    constructor() {
        this.run();
    }
    async run() {
        while (true) {
            var sentData = await port.getSentData();
            port.provideReceivedData(sentData);
        }
    }
}

function createDevice(): Port.IScriptDevice {
    return new LoopbackSerialDevicePull;
}
```

Receiving Data

Data model only governs the processing of the data sent by an application to a port. Whenever the device script needs to emulate received data, it calls the IPort.provideReceivedData method with the data it wants to make available for application. Next time application attempts to read from the serial port it gets a copy of those data.

Device Script API

Virtual script port provides the following functionality to the device script:

Port API

Provided by means of a global object `port` which implements the IPort interface. It represents the virtual serial port itself and provides methods for reading and writing data.

File System

Provided by means of a global object `fs` which implements the FS.IFileManager interface. It allows the device script to create and delete folders and files, open files for reading and writing and read and write data from them. Device script may also enumerate the contents of any folder.

Sockets

Provided by means of a global object `net` which implements the Net.INetworkManager interface. It allows the device script to create TCP and UDP sockets and use them to perform network communication.

HTTP

Provided by means of a global object `http` which implements the Http.IHttpClient interface. It allows the device script to send all kinds of HTTP requests.

Script Debugging

Device script runs in a controlled and secure environment. Therefore, external access to a running script is limited. The only debugging facility available for a device script is a global log method.

By default, all log files are created in a `%TEMP%\vspt_script_logs` folder. Note that `%TEMP%` environment variable is expanded under the context of a `System` account and is usually resolved into `c:\Windows\TEMP`. Log file folder location may also be changed by setting the IScriptPortDevice.logPath

property.

A log file is only created if a device script makes a call to a `log` method, or throws an unhandled exception. The log file name follows this scheme:

`vspt.YYYY-MM-dd.hh-mm-ss.PID.log`

Abbr	Description
`YYYY`	Year
`MM`	Month
`dd`	Day
`hh`	Hour in 24-hours format
`mm`	Minutes
`ss`	Seconds
`PID`	Process ID of a port opening process

Pipe Ports

Virtual Serial Port Tools allows connecting a virtual serial device to the named pipe. A serial device may either play the role of the named pipe server or named pipe client.

When acting as named pipe client, the other end of the pipe may reside either on the local or remote computer.

Connection

To connect a virtual serial device to the named pipe, create new virtual serial port with a call to ISerialPortLibrary.createPipePort method and use the IPipePortDevice.configureConnectPipe method to connect to an existing pipe (client mode) or IPipePortDevice.configureCreatePipe2 method to create a named pipe and connect to it in server mode.

The actual pipe creation or connection occurs only when a client application opens a handle to a virtual serial pipe.

Operation

When an application writes data to the serial port, all written data is transferred to the named pipe. Correspondingly, when application reads data from the port, the data is fetched from the named pipe.

Configuration Utility

Configuration Utility allows the user to create ports connected to pipes using the Create New Pipe Port Window.

Listening Ports

Listening port is a “server” side of a remote bridge. When you create a new remote bridge, a name of the remote listening virtual serial port needs to be specified.

Creation

To create new listening port, first create new virtual serial device using the

ISerialPortLibrary.createBridgePort method and then call its IBridgePortDevice.startListening method.

Operation

The listening port is initially disconnected, but it is ready to accept incoming connections to form a remote bridge. After connection, it becomes a remote bridge on the remote computer.

Configuration Utility

Configuration Utility allows the user to create listening ports using the Create Listening Port Window.

Sharing COM Ports over Network

Virtual Serial Port Tools includes two optional server components. They may be installed either among with other components or as stand-alone components. Both server components can be configured to run as Windows Services, or as stand-alone applications.

Remote Serial Ports Server

This component shares local COM ports over the network. It is best suited for local-area networks or domain networks. It provides the following features:

- Automatic advertising of shared COM ports.
- Fine-grained access control allows administrators to configure which users and groups have access to shared COM ports.
- Full emulation of underlying COM port is provided.
- Remote Ports are used on the "client" side to connect to shared serial ports.

The following topics provide more information on Remote Serial Ports Server component:

- Remote Serial Ports Server
- Server Configuration Utility
- Server Command-Line Parameters

TCP/IP Serial Ports Server

This component shares local COM ports over the TCP/IP network. It provides the following features:

- Support for RFC2217 standard serial port protocol.
- Support for RAW serial port protocol.
- Default port configuration parameters may be specified. They will be used if not overridden by applications.
- Full emulation of underlying COM port is provided.
- TCP/IP Ports are used on the "client" side to connect to shared serial ports.

The following topics provide more information on Remote Serial Ports Server component:

- TCP/IP Serial Ports Server
- TCP/IP Server Configuration Utility
- TCP/IP Server Command-Line Parameters

Remote Serial Ports Server

Remote Serial Ports Server is an optional component of Virtual Serial Port Tools. It should be installed on a computer which serial ports you want to share over the network. It can be installed either with other VSPT components or as a standalone component. By default, server is installed as Windows Service and runs even without a logged-on user. In addition, the server may be launched on-demand, providing a quick way to share serial devices.

Server component is used to provide remote access to all local serial devices, including legacy serial ports, virtual serial ports or "serial over USB" devices. The server can be used in one of the following modes: installed as Windows Service or running as a stand-alone process.

By default, all local serial devices are shared across the network and all users are granted access. In addition, the server automatically advertises itself on the local network.

All these defaults may be changed using either the command-line parameters if the server is running stand-alone mode or using the Server Configuration Utility if the server is installed as Windows Service. See below for more information.

Windows Service Mode

In this mode, server is installed as Windows Service and is configured to run without logged-on user. This is the default mode, configured by server installation utility. Server options are controlled with a help of Server Configuration Utility.

If the user needs to manually configure Remote Serial Ports Server to Windows Service mode, the following command-line parameter may be used:

```
Command Prompt
ps_server.exe -install-service
```

To remove Windows Service, use the following command-line parameter:

```
Command Prompt
ps_server.exe -uninstall-service
```

> **NOTE**
>
> Since installing or removing a Windows Service is a privileged operation, these command-lines must be executed from the elevated command-prompt.

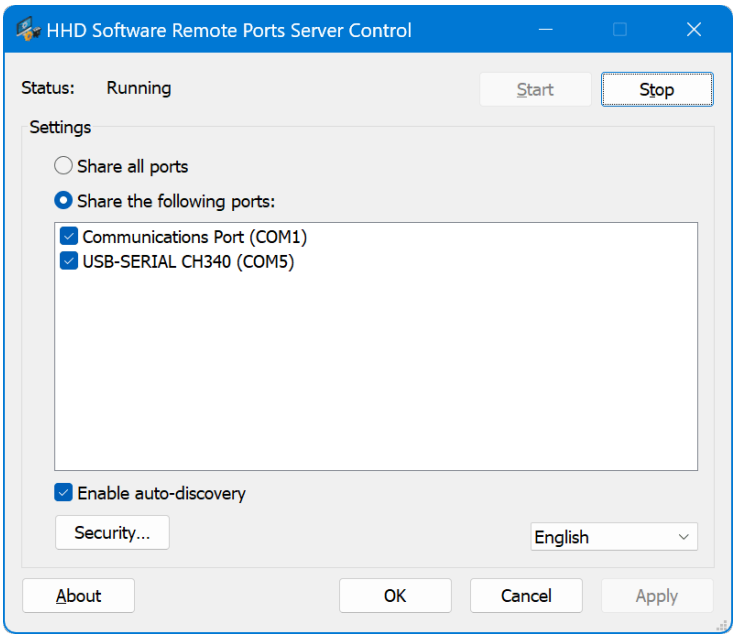
Stand-alone Mode

Remote Serial Ports Server supports simple deployment for quick serial sharing experience. All you need is to copy the `ps_server.exe` file to the target computer and launch it, optionally providing command-line parameters for fine-grain control. See the Server Command-Line Parameters section for more information.

Server Configuration Utility

Configuration utility (an optional component installed by Remote Serial Ports Server installation package) provides a way to configure Remote Serial Ports Server, running in Windows Service mode.

It may also be used to configure server running in stand-alone mode, however, the preferred way is to use command-line parameters.



At the top of the window you can see the current status of Windows Service. Use the **Start** and **Stop** buttons to control the service. If configuration utility is used to configure server running in stand-alone mode, these buttons are disabled.

Next, there's an option to select which ports are shared by the server. Default setting is to share all ports.

Check the “Enable auto-discovery” option to specify whether the server automatically advertises itself on the local network.

Pressing the **Security...** button brings up the Security window where you can configure which users and groups are granted access to shared serial ports. By default, all users are granted access.

When new settings are applied, server automatically restarts, loading new settings. All existing connections are kept.

Command-Line Parameters

When Remote Serial Ports Server is running as stand-alone process, use command-line parameters to configure its options. The server supports the following options:

Option	Argument	Description
`-?`, `--help`		Display command-line parameters.
`--nologo`		Do not display logo message.
Server options		
`--security-descriptor`	`SDDL`	Security descriptor in SDDL format
`--share-ports`	`N1[,N2[,N3...]]`	Only share specified ports.
`--no-discovery`		Turn automatic discovery off.
Logging options		
`--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 service. Must be called from elevated command-prompt.
`-uninstall-service`, `--uninstall-service`		Uninstall service. Must be called from elevated command-prompt.

TCP Serial Ports Server

TCP/IP Serial Ports Server is an optional component of Virtual Serial Port Tools. It needs to be installed on a computer which serial ports you want to expose on given TCP endpoints to be used by other computers on the network. By default, server is installed as Windows Service and runs even without a logged-on user. In addition, the server may be launched on-demand, providing quick way to share serial devices.

Server component is used to provide remote access to all local serial devices, including legacy serial ports, virtual serial ports or "serial over USB" devices.

> **NOTE**
>
> Current version always uses all available network interfaces when it listens for incoming TCP connections.

The server can be used in one of the following modes: installed as Windows Service or running as stand-alone process.

The server administrator has to explicitly specify which serial ports are shared on which TCP ports. When TCP/IP Serial Ports Server executes as a stand-alone process, this configuration is done using the TCP/IP Server Command-Line Parameters. When TCP/IP Serial Ports Server is running as Windows Service, configuration is performed using the TCP/IP Server Configuration Utility.

Windows Service Mode

In this mode, server is installed as Windows Service and is configured to run without logged-on user. This is the default mode, configured by server installation utility. Server options are controlled with a help of TCP/IP Server Configuration Utility.

If the user needs to manually configure TCP/IP Serial Ports Server to Windows Service mode, the following command-line parameter may be used:

```
Command Prompt
psip_server.exe -install-service
```

To remove Windows Service, use the following command-line parameter:

```
Command Prompt
psip_server.exe -uninstall-service
```

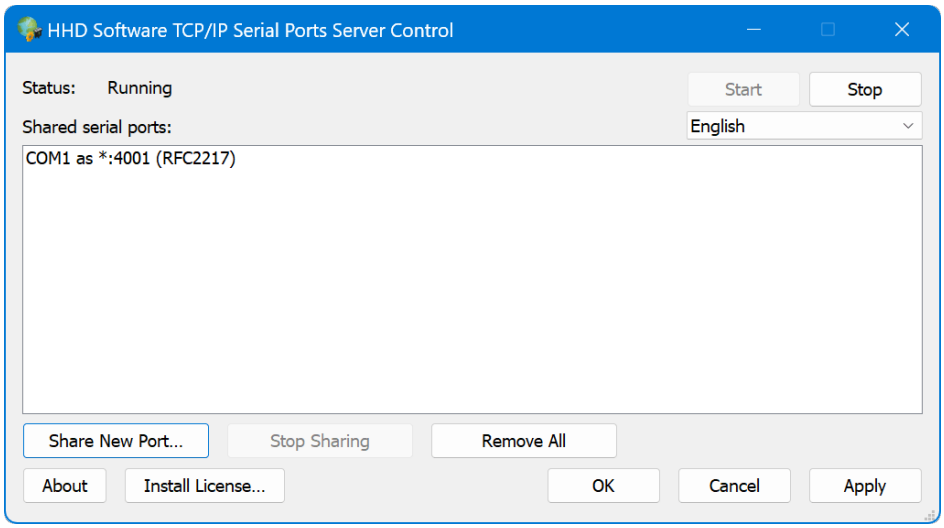
> **NOTE**
>
> Since installing or removing a Windows Service is a privileged operation, these command-lines must be executed from the elevated command-prompt.

Stand-alone Mode

TCP/IP Serial Ports Server supports simple deployment for quick serial sharing experience. All you need is to copy the `psip_server.exe` file to the target computer and launch it, providing command-line parameters to specify what serial ports and their corresponding TCP ports. See the TCP/IP Server Command-Line Parameters section for more information.

TCP/IP Server Configuration Utility

Configuration utility (an optional component installed by TCP/IP Serial Ports Server installation package) provides a way to configure TCP/IP Serial Ports Server, running in Windows Service mode.

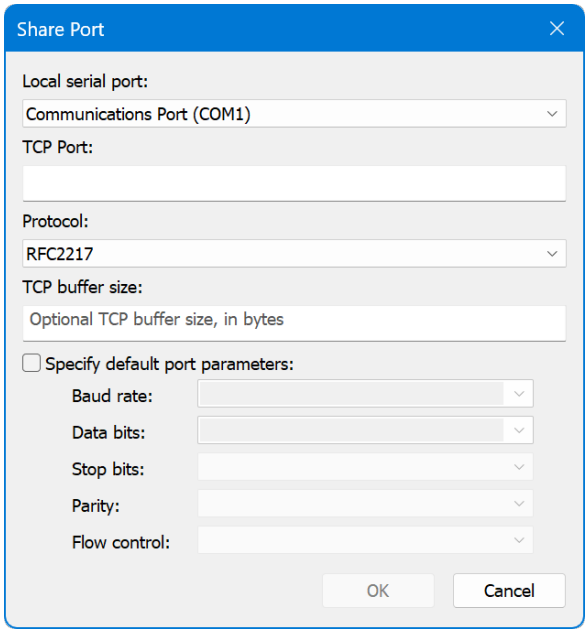


At the top of the window you can see the current status of Windows Service. Use the **Start** and **Stop** buttons to control the service.

Below is the list of currently shared ports. A single COM port may be shared over different TCP ports. In this case, only single connection will succeed, effectively locking a serial port.

> **NOTE**
>
> Current version always uses all available network interfaces when it listens for incoming TCP connections.

To share new serial port, press the **Share New Port...** button, select the COM port to share, enter TCP port number and choose the protocol.



To remove a share, select it in the list and press the **Stop Sharing** button. To remove all shared ports, press the **Remove All** button.

Press the **Apply** button to apply the changes. This will restart the server, closing all existing connections. Pressing **OK** button applies current settings and closes the application.

Command-Line Parameters

When TCP/IP Serial Ports Server is running as stand-alone process, use command-line parameters to configure its options. The server supports the following options:

```
Command Prompt
psip_server.exe [OPTIONS] <SHARED-PORTS>
```

Where `<SHARED-PORTS>` is encoded according to basic or advanced syntax:

Basic Syntax

```
Command Prompt
COMn1=TCP-PORT[,protocol] [COMn2=TCP-PORT[,protocol]]...
```

`COMn=<port>[,protocol]` arguments are mandatory. They specify which serial ports are shared over which TCP ports. `protocol`, if specified, must be either `rfc2217` or `raw`. If protocol is omitted, `rfc2217` is assumed.

Example:

```
Command Prompt
psip_server.exe COM5=11111 COM6=22222,raw
```

Advanced Syntax

```
Command Prompt
com=n,tcp=n[,comma-separated optional arguments]
```

Supported optional arguments:

Argument format	Description
`baud=n`	Override baud rate
`bits=n`	Override byte size
`stop=1|1.5|2`	Override stop bits
`parity=no|odd|even|mark|space`	Override parity
`flow=no|software|hardware`	Override flow control
`protocol=rfc2217|raw`	Override protocol
`bufferSize=n`	Set TCP buffer size, in bytes (default 512KB, allowed range 4KB..10MB)

Example:

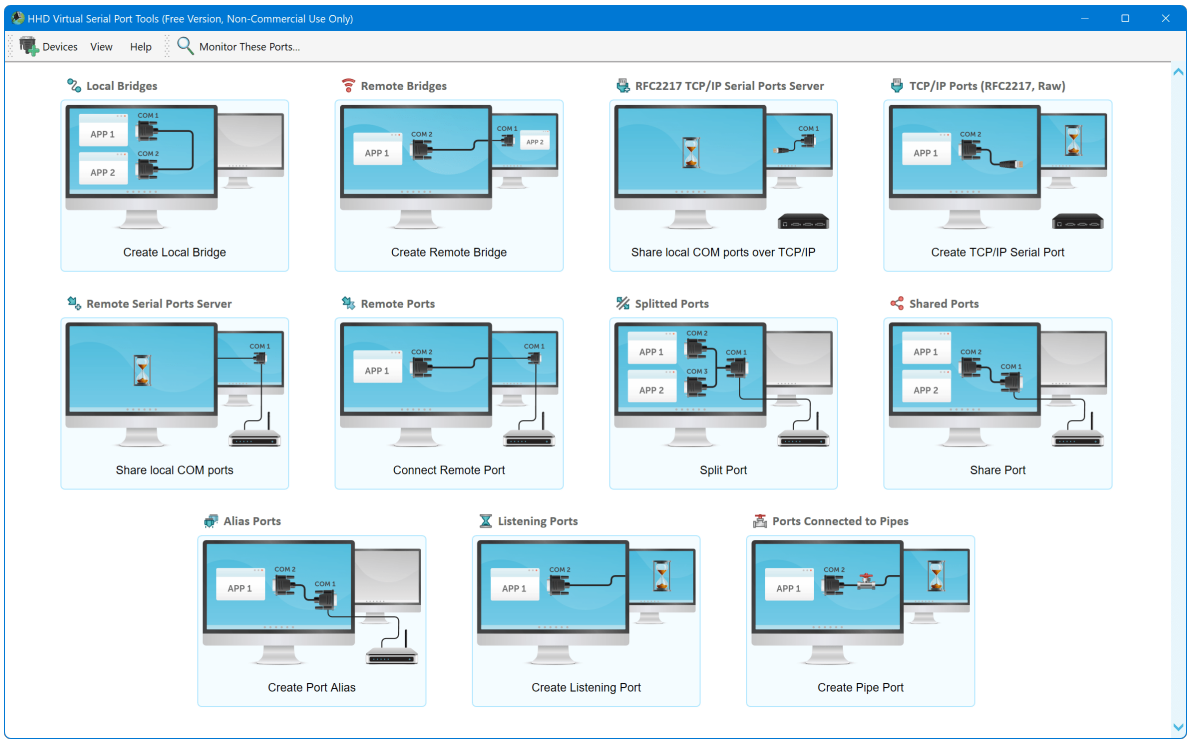
```
Command Prompt
psip_server.exe com=5,tcp=11111 com=6,tcp=22222,protocol=row,bufferSize=65536
```

Options

Option	Argument	Description
`-?`, `--help`		Display command-line parameters.
`--nologo`		Do not display logo message.
Logging options		
`--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 service. Must be called from elevated command-prompt.
`-uninstall-service`, `--uninstall-service`		Uninstall service. Must be called from elevated command-prompt.
License Installation		
`--license`	`<path>`	Install the license from the given license file

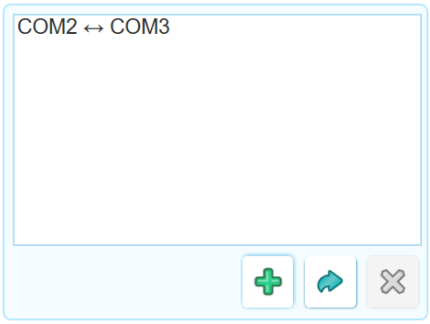
Configuration Utility

This application provides a graphical user interface to create and manage virtual serial devices and their configurations. It uses API provided by Virtual Serial Port Tools COM server.



Each supported device type is represented with a rich icon that describes the configuration this particular device type solves. Clicking on the icon opens the configuration window of a selected device type.

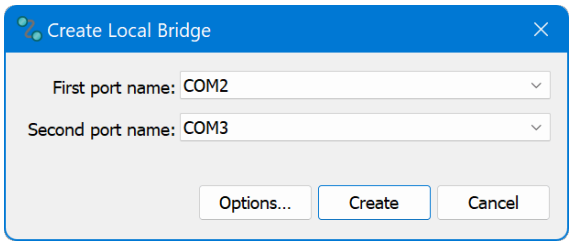
When at least one device of a given type is created, the icon is replaced with a device list:



Three buttons are: **Create**, **Export Configuration** and **Delete**. Pressing the first button opens the configuration window of a selected device type, pressing the second button opens the Exporting Configuration window and pressing the last button deletes the selected device.

Create Local Serial Bridges

This window allows you to create a new local virtual serial bridge:



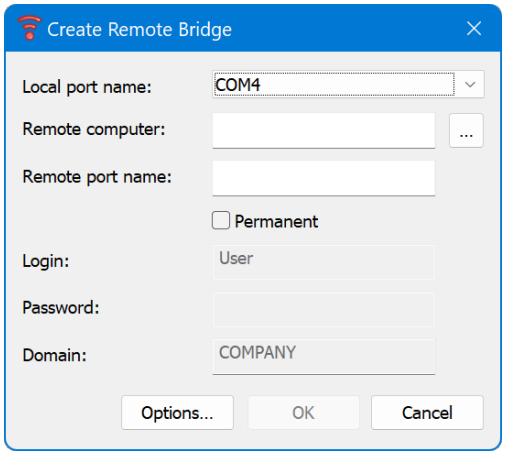
Here you configure port names of the first and second ports.

Bridge Creation Options

See the Bridge Creation Options topic for more information on customizing local bridge configuration.

Create Remote Serial Bridges

This wizard allows you to create new virtual serial device and connect it to remote virtual serial device, thus forming a virtual remote bridge.



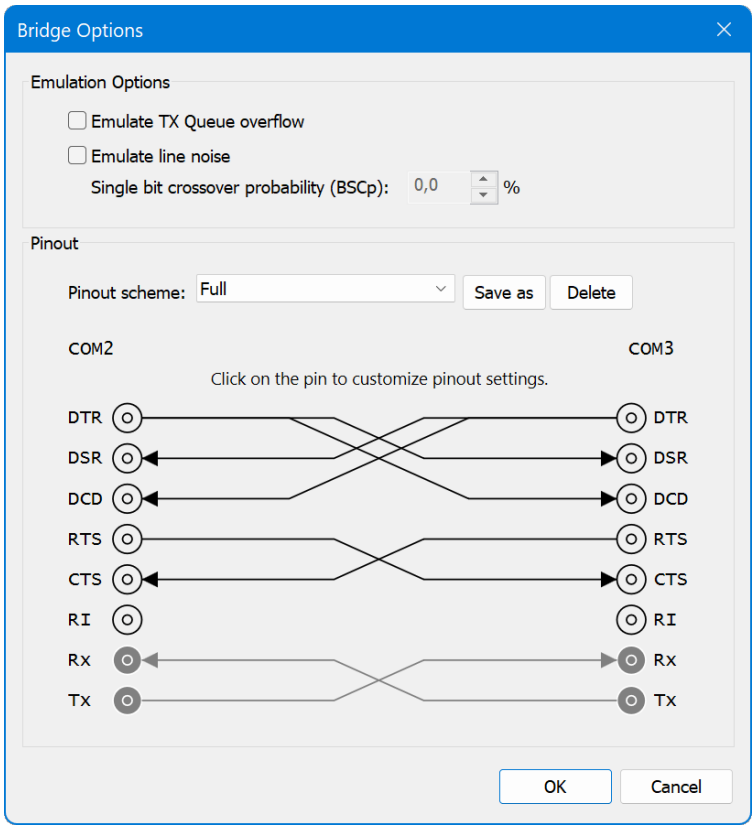
First, you need to select the name of the local port. Then, enter (or browse for) the name of the remote server and enter the name of the listening port on that server.

Bridge Creation Options

See the Bridge Creation Options topic for more information on customizing remote bridge configuration.

Bridge Creation Options

Bridge Options window allows you to enable baud rate emulation, overflow emulation, line noise emulation and custom pin-out.



By default, virtual serial bridge does not emulate baud rate. This behavior may be controlled by the **Emulate baud rate** setting.

By default, virtual serial bridge does not emulate overflow of transmit buffer. This behavior may be controlled by the **Emulate TX Queue Overflow** setting.

If you want to emulate line noise, check the "Emulate line noise" option and enter the single bit crossover probability (AKA BSCp), that is, a probability of flipping a single bit. Acceptable range is from 0.01% to 50.00%.

Pin-Out Configuration

If you need to apply a non-standard "wiring" configuration to the created bridge, expand the **Bridge Configuration** box. Then either select one of predefined or previously saved schemes using the *Pinout scheme* drop-down box, or use Drag&Drop to add new connections.

Click on one of the pins to see more information.

Selected pin has the brown color. Pins it may be connected to are drawn in green. Outgoing connections are painted blue while incoming connections are painted green.

Creating New Connection

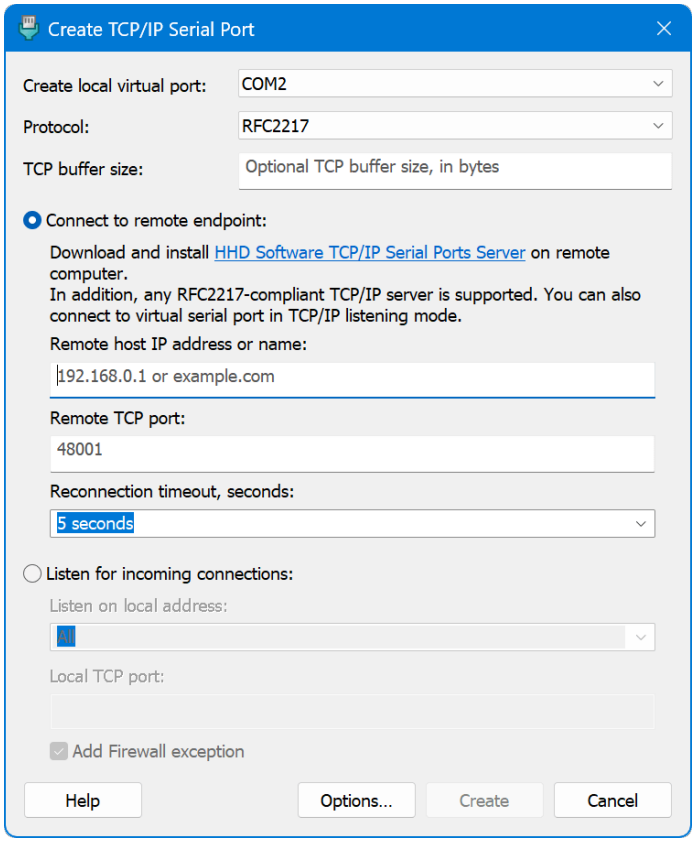
To create a new connection, either right-click the source pin and select the destination pin in a context menu or left-click and drag to the destination pin.

Deleting Connection

To remove a connection, either right-click the source pin and select the destination pin to unplug in a context menu; or left-click and drag to the destination pin to remove an existing connection.

Creating TCP Ports

This window allows you to create new virtual serial port and configure it.



Select the local port name you want to create or leave the default one. Select the protocol (currently, RFC2217 and RAW protocols are supported).

Creating TCP/IP Ports in Connecting Mode

To create a virtual serial port that connects to remote TCP/IP endpoint when opened by application, make sure the **Connect to remote endpoint** option is chosen. Then enter the remote host address or name, specify remote TCP port number and reconnection timeout, used to restore connection if the network connectivity error occurs during transmission.

Creating TCP/IP Ports in Listening Mode

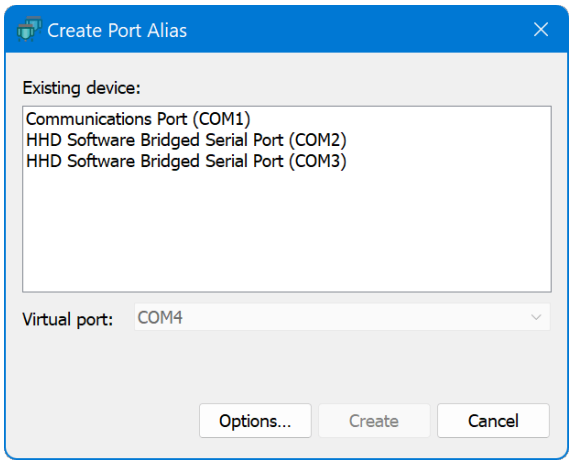
To create a virtual serial prot that will start listening on a specified local TCP/IP endpoint, make sure the **Listen for incoming connections** option is chosen. Then select which local address (or “All”) to use and specify local TCP port number.

Make sure the **Add Firewall exception** option is checked to automatically add a Firewall exception rule for incoming connection on specified port. This rule will automatically be removed if the virtual serial port is deleted.

Press the **Options...** button to set Port Settings Overrides options.

Create Alias/Mapped Serial Ports

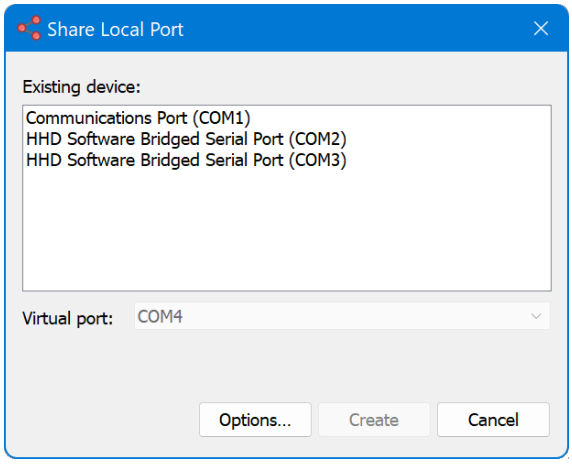
This window allows you to create a new alias port:



First select the original device for which you want to create an alias and then set the created port name. Optionally, specify the port settings overrides by pressing the **Options** button.

Create Shared Serial Ports

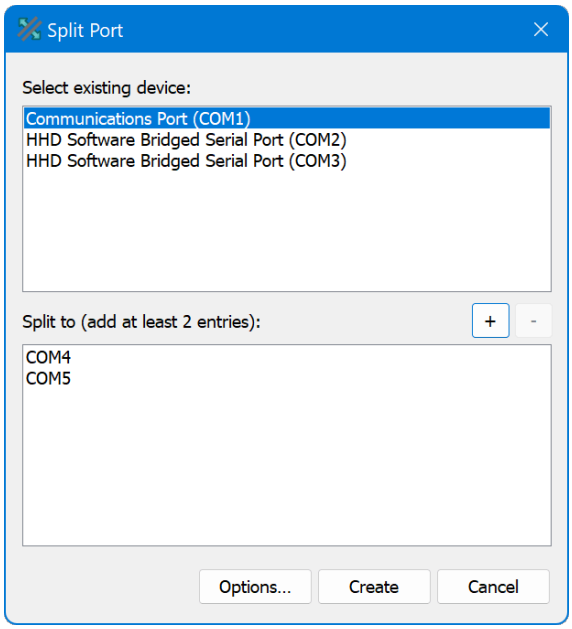
This window allows you to create a new shared port:



First select the original device which you want to share and then set the created port name. Optionally, specify the port settings overrides by pressing the **Options** button.

Create Split Serial Ports

This window allows you to create a split port configuration:

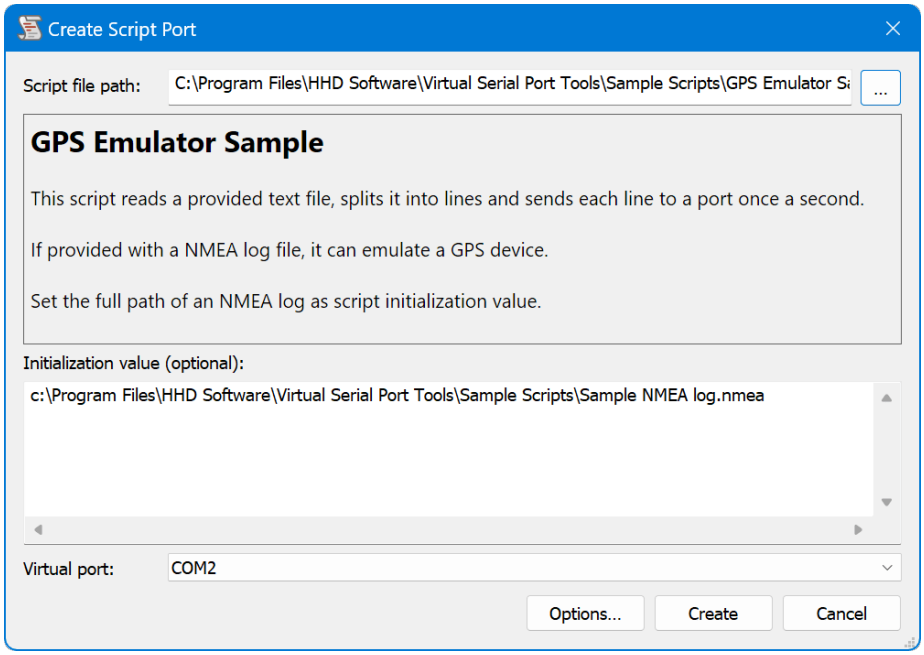


First select the device you want to split. Then press the **+** button to add two or more aliases for the port being split. You can also delete an added alias by selecting it in the list and pressing the **-** button.

Optionally, specify the port settings overrides by pressing the **Options** button.

Create Script Serial Ports

This window allows you to create a new script port:



Enter the full path to a device script or click the Browse button. If the script is one of the sample device scripts installed by Virtual Serial Port Tools installer, its short description will be displayed in the device creation window.

Optionally provide an initialization value and click the **Create** button.

The script file will be compiled and validated. If it passes the validation, a new virtual script device will be created, otherwise, a list of validation errors is displayed.

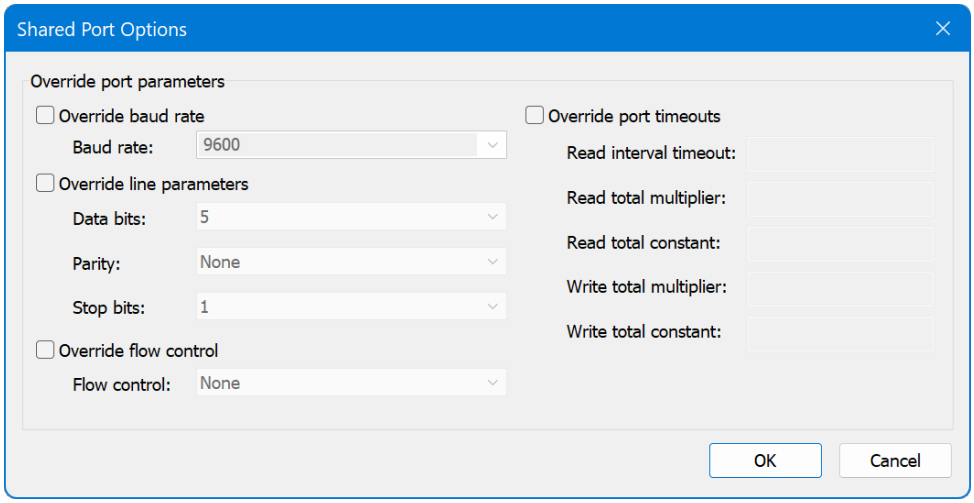
NOTE

For security reasons, once validation is successful, VSPT will never read the script file again. If you make changes to a device script file, make sure the script is updated, or virtual device is re-created in order for changes to be applied.

Optionally, specify the port settings overrides by pressing the **Options** button.

Port Settings Overrides

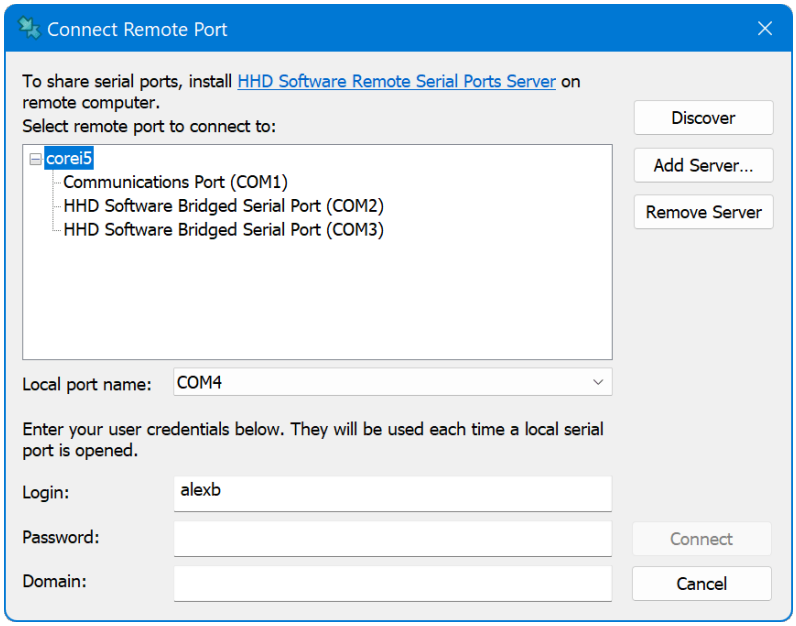
The port settings override window allows you to set specific port parameter overrides to be applied for a virtual port:



You can override the baud rate, line parameters, serial port timeouts and/or flow control settings.

Connect Remote Serial Port

This window allows you to create new virtual serial port that acts as a remote serial port.



Select the port in the server list. The server list is automatically populated with auto-discovered servers in the local area network or domain network. You can also add new server manually by pressing the **Add**

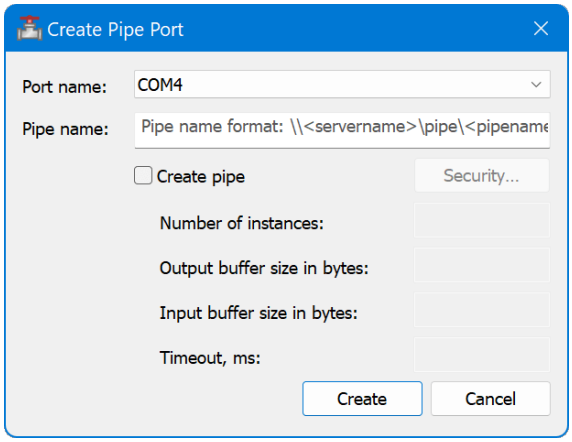
Server... button.

Below you can customize the created port name and enter the credentials to use for authentication on the remote server. Entered credentials are stored securely in the protected location associated with created serial device and are automatically persisted through system reboots.

> **NOTE**
>
> Although the actual connection (and authentication) occurs only when an application opens the created virtual serial device, configuration utility attempts to validate entered credentials by connecting to the remote computer. This attempt may fail or succeed using local user credentials even if entered credentials are invalid.
>
> However, when credentials are used later for actual connection, the process will execute under Local System account and entered credentials may actually result in connection failure.
>
> Make sure entered credentials are actually granted remote access to the specified server as well as granted access to shared ports on that server.

Create Pipe-connected Serial Ports

This wizard allows you to create new virtual serial device and connect it to a specified named pipe.



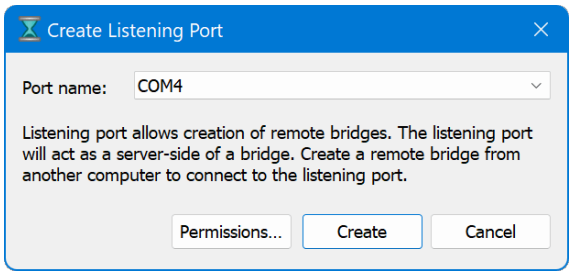
You can customize the created port name.

Pipe port is connected to a given named pipe, for which it is either a creator, or a client. First, specify the name of the pipe. Valid pipe names must conform to the following pattern: `\\servername\pipe\pipename`, where *servername* is either the name of the remote server or "." for local server. If you are the creator of the pipe, you must specify local server. *Pipename* must be any number of alphanumeric symbols and must not contain slash or backslash characters.

If you are creating a pipe, you may optionally specify pipe creation parameters.

Create Listening Serial Ports

This window allows you to create new listening serial port.

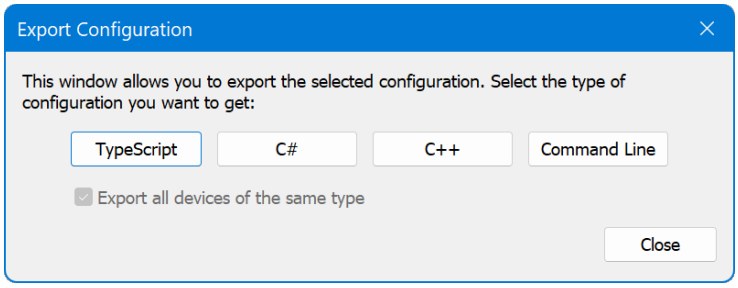


You can customize the created port name.

Listening port is initially disconnected, but is ready to accept remote connections to form remote bridges. If you select this port type, you may optionally configure listening port's access rights by pressing the **Permissions** button.

Exporting Configuration

This window allows you to export the selected configuration into one of supported formats.



You can choose one of the following formats for export:

TypeScript

The selected device configuration is exported as TypeScript code, for example, the following text is generated when you export a local bridge `COM3 ↔ COM4`:

```
TypeScript
///<reference path="hhdvspkit.d.ts" />
var library = (ISerialPortLibrary) new ActiveXObject("hhdvspkit.SerialPortLibrary.5");
{
    var port1 = library.createBridgePort(3);
    var port2 = library.createBridgePort(4);
    port1.bridgePort = 4;
    port2.bridgePort = 3;
}
```

C#

The selected device configuration is exported as C++ code. For example, the following text is generated when you export remote port `COM6`:

```
C#
var library = new hhdvspkit.SerialPortLibrary();
{
    var port = library.createRemotePort(6);
    port.remoteHost = "servername";
    port.remotePort = 1;
    port.login = "UserName";
    port.domain = "Domain";
    port.password = "<password>";
}
```

C++

The selected device configuration is exported as C++ code. For example, the following text is generated when you export an advanced configuration of a shared port `COM5`:

```C++
hhdvspkit::ISerialPortLibraryPtr library;
library.CreateInstance(__uuidof(hhdvspkit::SerialPortLibrary));
{
    auto port = library->createSharedPort(5);
    port->sharedPort = 1;
    port->baudRate = 115200;
    port->dataBits = 8;
    port->parity = hhdvspkit::PortParity::MarkParity;
    port->stopBits = hhdvspkit::PortStopBits::OneAndHalf;
    port->flowControl = hhdvspkit::PortFlowControl::Hardware;
}
```

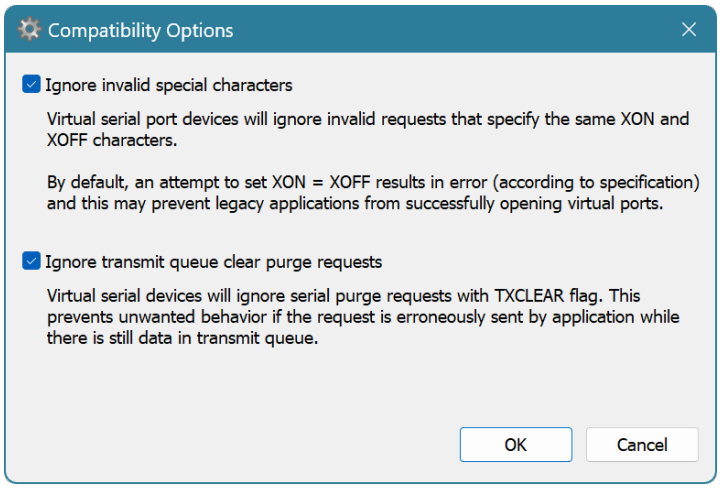
Command-Line

The selected device configuration is exported as one or more command-line text strings. For example, the following configuration creates and configures an alias port `COM7`:

```PowerShell
.\rspcli.exe -create alias --local-port 7 --alias-port 1 --baud-rate 57600 --flow-control
software
```

Compatibility Options

Virtual Serial Port Tools provides different compatibility options that can be configured to change the behavior of virtual serial port driver.



Currently, the following compatibility options are supported (see also ISerialPortLibrary.compatibilityFlags):

Value	Description
Ignore invalid special characters	Virtual serial ports must silently ignore invalid special characters set by application. By default, when this compatibility flag is not set, virtual serial ports fail requests that attempt to set invalid combination of special characters, as required by specification.
Ignore transmit queue clear purge requests	Virtual serial devices will ignore serial purge requests with TXCLEAR flag. This prevents unwanted behavior if the request is erroneously sent by application while there is still data in transmit queue. Currently applies to local bridges only.

Command-Line Utility

Command-line utility **rspcli.exe** may be used as a simple API to create and manage alias, shared, remote, pipe, tcp and script serial ports and local bridges. Command-line utility returns `0` if the requested operation is completed successfully, or non-zero error code (`HRESULT`). It also prints error description to `STDOUT` unless the `--silent` parameter is specified.

Command-line Parameters

The utility supports the following command-line parameters:

Parameter	Value	Descriptio
`-?, --help`		Displays the list of supported parameters with short description.
`--silent`		Do not display any error or success messages.
`-create`	`TYPE`	Create new virtual serial device. TYPE must be one of the following: **Type** — **Description** `alias` — alias port `bridge` — local or remote bridge `listening` — listening port `shared` — shared port `remote` — remote port `pipe` — pipe-connected port `tcp` — TCP port | `script` — script port
`-delete`	`N`	Delete an existing port.

Parameter	Value	Descriptio
`-list`	`TYPE`	List all existing virtual serial devices. TYPE must be one of the following:

Type	Description
`alias`	alias ports
`bridge`	local bridge
`remote-bridge`	remote bridges (client side)
`listening`	listening ports (server side of remote bridge)
`shared`	shared ports
`remote`	remote ports
`pipe`	pipe-connected ports
`tcp`	TCP ports
`script`	script ports

`-list-remote`	`hostname`	List all shared serial ports on the specified host.
**Port Creation Parameters*		
`--local-port`	`N`	Optional local port number. If omitted, the next available port is used.
`--device-name`	`name`	Optional name to use for created virtual serial port device.
Alias Port Parameters		
`--alias-port`	`N`	Original serial port number.
Bridge Parameters		

Parameter	Value	Descriptio
`--bridge-ports`	`N1,N2`	Optional bridges port numbers.
Shared Port Parameters		
`--share-port`	`N`	Original serial port number.
Pipe Port Parameters		
`--create-pipe`	`\\.\pipe\PIPENAME`	Name of the pipe to create.
`--connect-pipe`	`\\SERVERNAME\pipe\PIPENAME`	Name of the pipe to connect to.
`--num-instances`	`N`	Number of port instances, 255 to unlimited.
`--input-buffer-size`	`N`	Size of the input buffer.
`--output-buffer-size`	`N`	Size of the output buffer.
`--pipe-timeout`	`N`	Pipe timeout.
`--security-descriptor`	`string`	Security descriptor in SDDL format
Common Port Parameters		
`--baud-rate`	`N`	Specify port baud rate to override.
`--data-bits`	`N`	Specify port data bits to override.
`--parity`	`(no | odd | even | mark | space)`	Specify port parity to override.
`--stop-bits`	`(1 | 1.5 | 2)`	Specify port stop bits to override.
`--flow-control`	`(none | software | hardware)`	Specify port flow control to override.
`--timeouts`	`"N1,N2,N3,N4,N5"`	Specify port timeouts to override. Timeout values must be separated with commas (no spaces allowed) and must in the following order: readIntervalTimeout, readTotalMultiplier, readTotalConstant, writeTotalMultiplier, writeTotalConstant. -1 can be used to specify MAXDWORD. Enclose the whole parameter in double quotation marks if using negative numbers.
Remote Port Parameters		
`--remote-host`	`hostname`	Name or address of remote host that shares COM port.

Parameter	Value	Descriptio
`--remote-port`	`N`	Name of the remote COM port for RSP server or remote TCP port for TCP/IP server.
`--login`	`username`	Name of the user to use for authentication on a remote host. May include domain in the form `DOMAIN\USERNAME`.
`--password`	`password`	Password of the user for authentication on remote computer. If omitted, the user is asked to enter password in terminal.
`--connection-timeout`	`N`	An optional connection timeout (in milliseconds).
`--connection-attempts`	`N`	An optional number of connection attempts to try before giving up.
Listening Port Parameters		
`--security-descriptor`	`string`	Security descriptor in SDDL format.
Remote Bridge Parameters		
`--remote-host`	`hostname`	Name or address of remote host with server side of a remote bridge.
`--remote-port`	`N`	Port number of a listening port on a remote host.
`--login`	`username`	Name of the user to use for authentication on a remote host. May include domain in the form `DOMAIN\USERNAME`. Optional.
`--password`	`password`	Password of the user for authentication on remote computer. If omitted, the user is asked to enter password in terminal. Optional.
TCP Port Parameters		
`--tcp-protocol`	`(rfc2217 | raw)`	TCP/IP emulation protocol. Defaults to “rfc2217”. `rfc2217` RFC2217 protocol. `raw` Raw protocol.
`--local-address`	`(ip-address | *)`	Local address to listen on.

Parameter	Value	Descriptio
`--local-tcp-port`	`N`	Local TCP port number to listen on.
`--remote-host`	`hostname`	Name or address of remote host that shares COM port.
`--remote-port`	`N`	Remote TCP port number.
`--tcp-buffer-size`	`N`	TCP buffer size, in bytes. Default is 512KB. Allowed range: 4KB..10MB.
Script Port Parameters		
`--script-file`	`<path>`	Full or relative path to a script file.
`--script-log-folder`	`<path>`	Full path to a folder where script execution log files will be created.
`--script-init-value`	`string`	Any custom string to use for script initialization.
Compatibility Flags		
`--get-compat`	`N`	Gets the current compatibility flags. Can be a combination of the following values: `0` not set `1` ignore requests to set invalid special characters. `2` ignore transmit queue clear purge requests.
`--set-compat`	`N`	Sets the current compatibility flags. Can be a combination of the following values: `0` not set `1` ignore requests to set invalid special characters. `2` ignore transmit queue clear purge requests.

Redistribution

Virtual Serial Port Tools can be redistributed as part of another product providing the following conditions are met:

- A corresponding license with redistribution rights is obtained by a customer.
- VSPT installation package is not modified in any way and is still signed by a valid HHD Software Ltd. digital signature.

To facilitate simpler distribution, a separate redistributable package is provided on the product download page. It supports unattended installation and uninstallation and is a stripped-down version of a full installer with the following components removed:

- Configuration Utility
- Command-Line Utility
- Remote Serial Ports Server Component
- TCP/IP Serial Ports Server Component
- Documentation
- API headers and definitions

Installer Command-Line

Unattended Installation

```PowerShell
.\virtual-serial-port-tools-redist.exe -silent
```

Unattended Uninstallation

```PowerShell
.\virtual-serial-port-tools-redist.exe -silent -u
```

A calling process must have “Install Driver” privilege and must be elevated. Installer returns an integer which is zero when successful, equals `S_FALSE` if successful, but restart is required, or otherwise should be interpreted as `HRESULT` error code.

License Installation

The library must be activated on the client computer after installation. The user is supposed to activate the library using the license file embedded in the product installer. The license activation is performed with a call to ISerialPortLibrary.installLicenseFile or ISerialPortLibrary.installLicenseInMemory methods.

The following scenarios are supported:

- The library is activated by the product installer after installing VSPT redistributable package.
- The library is activated when the product is started for the first time.
- The library is activated each time the product is started.

Therefore, it is safe to activate the library multiple times.

Activation is per-user. If the VSPT is supposed to be used by multiple Windows users after installation, the product must activate the library for each Windows user.

Server Components Redistribution

If required, both Remote Serial Ports Server and TCP/IP Serial Ports Server components may be redistributed separately (without the need to install the full package). Both components are single

executable files (`ps_server.exe` and `psip_server.exe` correspondingly) without any dependencies.

Servers support simple "Copy&Paste" style deployment. They can be launched from non-elevated command prompt (or PowerShell or Windows Terminal) window and configured using command-line parameters.

In addition, both servers can be configured to be run as Windows Services. In this case, their corresponding configuration utilities must be used to configure them.

NOTE

TCP/IP Serial Ports Server requires the license to be installed on the computer it is running on.

VSPT API

Virtual Serial Port Tools installs an in-process COM server providing full-featured API to native, .NET or scripting clients. This documentation section describes the provided API in detail.

Any user code that intends to call any API method or use any API interface must have sufficient rights on the local computer. This means at least `SeLoadDriverPrivilege` privilege must be granted to the calling process and usually also means that the caller needs to be running elevated.

Using from Native Code

In order to use the API in native code, include the supplied header files:

```
C++
#include "clientctl.tlh"
#include "clientctl.tli"
```

These files are pre-installed in the `%INSTALLDIR%\api\native` folder.

After that you can use all library classes and interfaces. For example, to create an instance of the library object and then create a new alias virtual serial port, use the following code:

```
C++
hhdvspkit::ISerialPortLibraryPtr pLibrary;
if (SUCCEEDED(pLibrary.CreateInstance(__uuidof(hhdvspkit::SerialPortLibrary))))
{
      // create new alias virtual serial port with automatically-assigned port name
      auto pNewPort = pLibrary->createAliasPort();
      // configure created port
      pNewPort1->put_aliasPort = 1;
      // ...
}
```

Using from C#

In order to use the library from C# project, add a reference to the supplied `hhdvspkit.interop.dll` file to your project in Microsoft Visual Studio. The file is pre-installed in the `%INSTALLDIR%\api\interop` folder.

Use the API provided by the library:

```
C#
var library = new hhdvspkit.SerialPortLibrary();
var port = library.createAliasPort();
port.aliasPort = 1;
// ...
```

Adding a reference to `hhdvspkit.dll` in Visual Studio generates an interop DLL that exposes .NET wrappers for native code provided by the COM server. You can also add a reference to the included `hhdvspkit.interop.dll`, which is a pre-generated version of the interop DLL.

Using from JavaScript

To create an instance of the library, use the following code:

```
JavaScript
var library = new ActiveXObject("hhdvspkit.SerialPortLibrary.5");
```

Then, use the library object to create and manage virtual serial devices:

```JavaScript
// create new virtual serial port
var port = library.createAliasPort();
// configure serial port
port.aliasPort = 1;
// ...
```

Using from TypeScript

Virtual Serial Port Tools library contains TypeScript definition file that simplifies library usage in scripting environments. It automatically gives you method parameter type validation and simplifies specifying callback methods.

```TypeScript
///<reference path="hhdvspkit.d.ts" />
var library = (ISerialPortLibrary) new ActiveXObject("hhdvspkit.SerialPortLibrary.5");
// create new virtual serial port with automatically-assigned port name
var port = library.createAliasPort();
// configure port
port.aliasPort = 1;
// ...
```

ISerialPortLibrary

ISerialPortLibrary Interface

```TypeScript
interface ISerialPortLibrary {
    // Properties
    compatibilityFlags: CompatibilityFlags;

    // Methods
    createAliasPort(port?: number | string, deviceName?: string): IAliasPortDevice;
    createBridgePort(port?: number | string, deviceName?: string): IBridgePortDevice;
    createSharedPort(port?: number | string, reserved?: boolean, deviceName?: string):
ISharedPortDevice;
    createScriptPort(port?: number | string, deviceName?: string): IScriptPortDevice;
    createTcpPort(port?: number | string, deviceName?: string): ITcpPortDevice;
    createRemotePort(port?: number | string, deviceName?: string): IRemotePortDevice;
    createPipePort(port?: number | string, deviceName?: string): IPipePortDevice;
    getRemoteSharedPortsJs(hostName: string): IRemotePortDescription[];
    getPortsJs(type: SerialPortType): IDevice[];
    getPortsJs(type: SerialPortType.Remote): IRemotePortDevice[];
    getPortsJs(type: SerialPortType.Shared): ISharedPortDevice[];
    getPortsJs(type: SerialPortType.Alias): IAliasPortDevice[];
    getPortsJs(type: SerialPortType.Bridge): IBridgePortDevice[];
    getPortsJs(type: SerialPortType.Pipe): IPipePortDevice[];
    getPortsJs(type: SerialPortType.Tcp): ITcpPortDevice[];
    getPortsJs(type: SerialPortType.Script): IScriptPortDevice[];
    createTimeoutsObject(readIntervalTimeout: number,
        readTotalTimeoutMultiplier: number,
        readTotalTimeoutConstant: number,
        writeTotalTimeoutMultiplier: number,
        writeTotalTimeoutConstant: number): ITimeouts;
    installLicenseFile(path: string): void;
}
```

```C#
public interface ISerialPortLibrary
{
    // Properties
    CompatibilityFlags compatibilityFlags { get; set; }

    // Methods
    IAliasPortDevice createAliasPort(object port, string deviceName);
    IBridgePortDevice createBridgePort(object port, string deviceName);
    ISharedPortDevice createSharedPort(object port, bool reserved, string deviceName);
    IScriptPortDevice createScriptPort(object port, string deviceName);
    ITcpPortDevice createTcpPort(object port, string deviceName);
    IRemotePortDevice createRemotePort(object port, string deviceName);
    IPipePortDevice createPipePort(object port, string deviceName);
    Array getRemoteSharedPorts(string hostName);
    Array getPorts(SerialPortType type);
    ITimeouts createTimeoutsObject(uint readIntervalTimeout,
        uint readTotalTimeoutMultiplier,
        uint readTotalTimeoutConstant,
        uint writeTotalTimeoutMultiplier,
        uint writeTotalTimeoutConstant);
    void addListener(ISerialPortLibraryListener pListener);
    void removeListener(ISerialPortLibraryListener pListener);
    void installLicenseFile(string path);
    void installLicenseInMemory(byte[] data);
}
```

```C++
struct ISerialPortLibrary : IDispatch
{
    // Properties
    CompatibilityFlags compatibilityFlags;  // get set

    // Methods
    IAliasPortDevicePtr createAliasPort(const _variant_t & port, VARIANT deviceName);
    IBridgePortDevicePtr createBridgePort(const _variant_t & port, VARIANT deviceName);
    ISharedPortDevicePtr createSharedPort(const _variant_t & port, VARIANT reserved, VARIANT
deviceName);
    IScriptPortDevicePtr createScriptPort(const _variant_t & port, VARIANT deviceName);
    ITcpPortDevicePtr createTcpPort(const _variant_t & port, VARIANT deviceName);
    IRemotePortDevicePtr createRemotePort(const _variant_t & port, VARIANT deviceName);
    IPipePortDevicePtr createPipePort(const _variant_t & port, VARIANT deviceName);
    SAFEARRAY(IRemotePortDescription) getRemoteSharedPorts(_bstr_t hostName);
    SAFEARRAY(IDevice) getPorts(SerialPortType type);
    ITimeoutsPtr createTimeoutsObject(unsigned readIntervalTimeout,
        unsigned readTotalTimeoutMultiplier,
        unsigned readTotalTimeoutConstant,
        unsigned writeTotalTimeoutMultiplier,
        unsigned writeTotalTimeoutConstant);
    HRESULT addListener(ISerialPortLibraryListener * pListener);
    HRESULT removeListener(ISerialPortLibraryListener * pListener);
    HRESULT installLicenseFile(_bstr_t path);
    HRESULT installLicenseInMemory(SAFEARRAY(BYTE) data);
};
```

ISerialPortLibrary Properties

compatibilityFlags

```TypeScript
compatibilityFlags: CompatibilityFlags;
```

```C#
CompatibilityFlags compatibilityFlags { get; set; }
```

```
C++
CompatibilityFlags compatibilityFlags;  // get set
```

Description

Zero or more of compatibility flags that can be set to increase compatibility with legacy software.

ISerialPortLibrary Methods

createAliasPort

```
TypeScript
createAliasPort(port?: number | string, deviceName?: string): IAliasPortDevice;
```

```
C#
IAliasPortDevice createAliasPort(object port, string deviceName);
```

```
C++
IAliasPortDevicePtr createAliasPort(const _variant_t & port, VARIANT deviceName);
```

Parameters

`port`

Port number or a string formatted as `COMn`. If omitted, a next available port number is automatically selected.

`deviceName`

Optional device name (visible in Device Manager). By default, "HHD Software Alias Serial Port" is used. In native code, variant type must be `VT_EMPTY` or `VT_BSTR`.

Return Value

The method returns a reference to created port device.

Description

Create new alias virtual serial port.

createBridgePort

```
TypeScript
createBridgePort(port?: number | string, deviceName?: string): IBridgePortDevice;
```

```
C#
IBridgePortDevice createBridgePort(object port, string deviceName);
```

```
C++
IBridgePortDevicePtr createBridgePort(const _variant_t & port, VARIANT deviceName);
```

Parameters

`port`

Port number or a string formatted as `COMn`. If omitted, a next available port number is

automatically selected.

`deviceName`

Optional device name (visible in Device Manager). By default, "HHD Software Bridged Serial Port" is used. In native code, variant type must be `VT_EMPTY` or `VT_BSTR`.

Return Value

The method returns a reference to created port device.

Description

Create new virtual serial port that will act as one of the local bridge sides.

---

createSharedPort

```
TypeScript
createSharedPort(port?: number | string, reserved?: boolean, deviceName?: string):
ISharedPortDevice;
```

```
C#
ISharedPortDevice createSharedPort(object port, bool reserved, string deviceName);
```

```
C++
ISharedPortDevicePtr createSharedPort(const _variant_t & port, VARIANT reserved, VARIANT
deviceName);
```

Parameters

`port`

Port number or a string formatted as `COMn`. If omitted, a next available port number is automatically selected.

`reserved`

This parameter is reserved, you must pass `false` to it, or omit entirely.

`deviceName`

Optional device name (visible in Device Manager). By default, "HHD Software Shared Serial Port" is used. In native code, variant type must be `VT_EMPTY` or `VT_BSTR`.

Return Value

The method returns a reference to created port device.

Description

Create new shared virtual serial port.

---

createScriptPort

```
TypeScript
createScriptPort(port?: number | string, deviceName?: string): IScriptPortDevice;
```

```
C#
IScriptPortDevice createScriptPort(object port, string deviceName);
```

```
C++
IScriptPortDevicePtr createScriptPort(const _variant_t & port, VARIANT deviceName);
```

Parameters

`port`

Port number or a string formatted as `COMn`. If omitted, a next available port number is automatically selected.

`deviceName`

Optional device name (visible in Device Manager). By default, "HHD Software Script Serial Port" is used. In native code, variant type must be `VT_EMPTY` or `VT_BSTR`.

Return Value

The method returns a reference to created port device.

Description

Create new virtual script serial port.

---

createTcpPort

```
TypeScript
createTcpPort(port?: number | string, deviceName?: string): ITcpPortDevice;
```

```
C#
ITcpPortDevice createTcpPort(object port, string deviceName);
```

```
C++
ITcpPortDevicePtr createTcpPort(const _variant_t & port, VARIANT deviceName);
```

Parameters

`port`

Port number or a string formatted as `COMn`. If omitted, a next available port number is automatically selected.

`deviceName`

Optional device name (visible in Device Manager). By default, "HHD Software Network Serial Port" is used. In native code, variant type must be `VT_EMPTY` or `VT_BSTR`.

Return Value

The method returns a reference to created port device.

Description

Create new TCP virtual serial port.

---

createRemotePort

```
TypeScript
createRemotePort(port?: number | string, deviceName?: string): IRemotePortDevice;
```

```C#
IRemotePortDevice createRemotePort(object port, string deviceName);
```

```C++
IRemotePortDevicePtr createRemotePort(const _variant_t & port, VARIANT deviceName);
```

Parameters

`port`

Port number or a string formatted as `COMn`. If omitted, a next available port number is automatically selected.

`deviceName`

Optional device name (visible in Device Manager). By default, "HHD Software Remote Serial Port" is used. In native code, variant type must be `VT_EMPTY` or `VT_BSTR`.

Return Value

The method returns a reference to created port device.

Description

Create new remote virtual serial port.

---

createPipePort

```TypeScript
createPipePort(port?: number | string, deviceName?: string): IPipePortDevice;
```

```C#
IPipePortDevice createPipePort(object port, string deviceName);
```

```C++
IPipePortDevicePtr createPipePort(const _variant_t & port, VARIANT deviceName);
```

Parameters

`port`

Port number or a string formatted as `COMn`. If omitted, a next available port number is automatically selected.

`deviceName`

Optional device name (visible in Device Manager). By default, "HHD Software Pipe Serial Port" is used. In native code, variant type must be `VT_EMPTY` or `VT_BSTR`.

Description

Create new pipe virtual serial port.

---

getRemoteSharedPorts

```TypeScript
// This method is not available in scripting environment
```

```
C#
Array getRemoteSharedPorts(string hostName);
```

```
C++
SAFEARRAY(IRemotePortDescription) getRemoteSharedPorts(_bstr_t hostName);
```

Parameters

`hostName`

Name of the remote host which ports you want to enumerate.

Return Value

Method returns an array of IRemotePortDescription objects containing all shared ports on a remote server.

Description

Get a list of all shared serial ports on a remote server specified by `hostname` parameter.

---

getRemoteSharedPortsJs

```
TypeScript
getRemoteSharedPortsJs(hostName: string): IRemotePortDescription[];
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`hostName`

Name of the remote host which ports you want to enumerate.

Return Value

Method returns an array of IRemotePortDescription objects containing all shared ports on a remote server.

Description

Get a list of all shared serial ports on a remote server specified by `hostname` parameter.

---

getPorts

```
TypeScript
// This method is not available in scripting environment
```

```
C#
Array getPorts(SerialPortType type);
```

```C++
SAFEARRAY(IDevice) getPorts(SerialPortType type);
```

Parameters

`type`

The type of port devices to return.

Return Value

This method returns an array of all local virtual serial port devices of specified type.

---

getPortsJs

```TypeScript
getPortsJs(type: SerialPortType): IDevice[];
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`type`

A type of virtual serial port to return.

Return Value

This method returns an array of all local virtual serial port devices of specified type.

---

getPortsJs

```TypeScript
getPortsJs(type: SerialPortType.Remote): IRemotePortDevice[];
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`type`

A remote virtual serial port type.

Return Value

This method returns an array of all remote virtual serial port devices.

---

getPortsJs

```TypeScript
getPortsJs(type: SerialPortType.Shared): ISharedPortDevice[];
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`type`

A shared virtual serial port type.

Return Value

This method returns an array of all shared virtual serial port devices.

---

getPortsJs

```TypeScript
getPortsJs(type: SerialPortType.Alias): IAliasPortDevice[];
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`type`

A alias virtual serial port type.

Return Value

This method returns an array of all alias virtual serial port devices.

---

getPortsJs

```TypeScript
getPortsJs(type: SerialPortType.Bridge): IBridgePortDevice[];
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`type`

A bridge virtual serial port type.

Return Value

This method returns an array of all bridge virtual serial port devices.

---

getPortsJs

```TypeScript
getPortsJs(type: SerialPortType.Pipe): IPipePortDevice[];
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`type`

A pipe virtual serial port type.

Return Value

This method returns an array of all pipe virtual serial port devices.

---

getPortsJs

```TypeScript
getPortsJs(type: SerialPortType.Tcp): ITcpPortDevice[];
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`type`

A TCP/IP virtual serial port type.

Return Value

This method returns an array of all TCP/IP virtual serial port devices.

---

getPortsJs

```TypeScript
getPortsJs(type: SerialPortType.Script): IScriptPortDevice[];
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`type`

A script virtual serial port type.

Return Value

This method returns an array of all script virtual serial port devices.

---

createTimeoutsObject

```TypeScript
createTimeoutsObject(readIntervalTimeout: number,
    readTotalTimeoutMultiplier: number,
    readTotalTimeoutConstant: number,
    writeTotalTimeoutMultiplier: number,
    writeTotalTimeoutConstant: number): ITimeouts;
```

```C#
ITimeouts createTimeoutsObject(uint readIntervalTimeout,
    uint readTotalTimeoutMultiplier,
    uint readTotalTimeoutConstant,
    uint writeTotalTimeoutMultiplier,
    uint writeTotalTimeoutConstant);
```

```C++
ITimeoutsPtr createTimeoutsObject(unsigned readIntervalTimeout,
    unsigned readTotalTimeoutMultiplier,
    unsigned readTotalTimeoutConstant,
    unsigned writeTotalTimeoutMultiplier,
    unsigned writeTotalTimeoutConstant);
```

Parameters

`readIntervalTimeout`

Read interval timeout.

`readTotalTimeoutMultiplier`

Read total timeout multiplier.

`readTotalTimeoutConstant`

Read total timeout constant.

`writeTotalTimeoutMultiplier`

Write total timeout multiplier.

`writeTotalTimeoutConstant`

Write total timeout constant.

Description

This method creates a timeouts override object. Scripting languages may construct and assign timeouts override objects directly, while native and managed languages may use this helper method to create a pre-filled timeouts override object.

addListener

```TypeScript
// This method is not available in scripting environment
```

```C#
void addListener(ISerialPortLibraryListener pListener);
```

```C++
HRESULT addListener(ISerialPortLibraryListener * pListener);
```

Parameters

`pListener`

Pointer or reference to the listener interface implemented by the caller.

Description

Adds specified listener object to the list of listener objects. When new virtual serial devices are created or deleted, library calls methods of the supplied listener object, passing the list of affected devices.

---

removeListener

```TypeScript
// This method is not available in scripting environment
```

```C#
void removeListener(ISerialPortLibraryListener pListener);
```

```C++
HRESULT removeListener(ISerialPortLibraryListener * pListener);
```

Parameters

`pListener`

Pointer or reference to the listener interface implemented by the caller.

Description

Removes specified listener object from the list of listener objects.

---

installLicenseFile

```TypeScript
installLicenseFile(path: string): void;
```

```C#
void installLicenseFile(string path);
```

```C++
HRESULT installLicenseFile(_bstr_t path);
```

Parameters

`path`

A full path to license file.

Description

Install the given license file. Throws an exception if an error occurs.

---

installLicenseInMemory

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void installLicenseInMemory(byte[] data);
```

```
C++
HRESULT installLicenseInMemory(SAFEARRAY(BYTE) data);
```

Parameters

`data`

License data in memory.

Description

Install the license from the memory buffer.

ISerialPortLibraryListener Interface

Description

This interface is implemented by the user application to receive notifications about created and deleted virtual serial devices. An instance of the object implementing this interface must be registered with a library by calling ISerialPortLibrary.addListener method.

Declaration

```
TypeScript
// This interface is not available in scripting environment
```

```
C#
public interface ISerialPortLibraryListener
{

    // Methods
    void added(SerialPortType type, Array devices);
    void deleted(SerialPortType type, Array devices);
}
```

```C++
struct ISerialPortLibraryListener : IDispatch
{

    // Methods
    HRESULT added(SerialPortType type, SAFEARRAY(IDevice) devices);
    HRESULT deleted(SerialPortType type, SAFEARRAY(IDevice) devices);
};
```

ISerialPortLibraryListener Methods

added

```TypeScript
// This method is not available in scripting environment
```

```C#
void added(SerialPortType type, Array devices);
```

```C++
HRESULT added(SerialPortType type, SAFEARRAY(IDevice) devices);
```

Parameters

`type`

The type of devices in the notification

`devices`

Array of devices that are being created.

Description

Called when devices of specified type are being created.

deleted

```TypeScript
// This method is not available in scripting environment
```

```C#
void deleted(SerialPortType type, Array devices);
```

```C++
HRESULT deleted(SerialPortType type, SAFEARRAY(IDevice) devices);
```

Parameters

`type`

The type of devices in the notification

`devices`

Array of devices that have been deleted.

Description

Called after devices of specified type have been deleted.

SerialPortType

IDevice Interface

Description

This is a base interface for all supported device types: alias, shared, remote and local bridge.

Declaration

```
TypeScript
interface IDevice {
    // Properties
    readonly port: number;
    readonly devicePath: string;
    readonly openingInfo: IOpeningInfo;

    // Methods
    deleteDevice(): void;
}
```

```
C#
public interface IDevice
{
    // Properties
    uint port { get; }
    string devicePath { get; }
    IOpeningInfo openingInfo { get; }

    // Methods
    void deleteDevice();
}
```

```
C++
struct IDevice : IDispatch
{
    // Properties
    unsigned port;  // get
    _bstr_t devicePath;  // get
    IOpeningInfo openingInfo;  // get

    // Methods
    HRESULT deleteDevice();
};
```

IDevice Properties

port

```
TypeScript
readonly port: number;
```

```
C#
uint port { get; }
```

```
C++
unsigned port;  // get
```

Description

Contains the port number assigned by the OS to the current virtual serial port device.

---

devicePath

```
TypeScript
readonly devicePath: string;
```

```
C#
string devicePath { get; }
```

```
C++
_bstr_t devicePath;  // get
```

Description

Contains the unique port path. This unique path may be passed to `CreateFile` function instead of usual `COMn`.

---

openingInfo

```
TypeScript
readonly openingInfo: IOpeningInfo;
```

```
C#
IOpeningInfo openingInfo { get; }
```

```
C++
IOpeningInfo openingInfo;  // get
```

Description

An object whose properties can be queried to get the information about the process that opened the port and current port parameters.

If the port is not currently opened, an exception is thrown.

IDevice Methods

---

deleteDevice

```
TypeScript
deleteDevice(): void;
```

```
C#
void deleteDevice();
```

```
C++
HRESULT deleteDevice();
```

Description

Call this method to delete the virtual serial port device. Actual device is deleted immediately (notifying

any registered listeners through their ISerialPortLibraryListener.deleted method), however, the API device object is only destroyed when all references are released.

ISerialPortLibrary.getPorts and ISerialPortLibrary.getPortsJs methods do not list deleted devices, even if there are still references to them.

IConfigurableDevice Interface

Description

This is a base interface, that derives from IDevice and supports alias and shared ports.

Declaration

```
TypeScript
interface IConfigurableDevice extends IDevice {
    // Properties
    baudRate: number;
    dataBits: number;
    parity: PortParity;
    stopBits: PortStopBits;
    flowControl: PortFlowControl;
    timeouts: ITimeouts;
}
```

```
C#
public interface IConfigurableDevice : IDevice
{
    // Properties
    uint baudRate { get; set; }
    uint dataBits { get; set; }
    PortParity parity { get; set; }
    PortStopBits stopBits { get; set; }
    PortFlowControl flowControl { get; set; }
    ITimeouts timeouts { get; set; }
}
```

```
C++
struct IConfigurableDevice : IDevice
{
    // Properties
    unsigned baudRate;  // get set
    unsigned dataBits;  // get set
    PortParity parity;  // get set
    PortStopBits stopBits;  // get set
    PortFlowControl flowControl;  // get set
    ITimeoutsPtr timeouts;  // get set
};
```

IConfigurableDevice Properties

baudRate

```
TypeScript
baudRate: number;
```

```
C#
uint baudRate { get; set; }
```

```cpp
C++
unsigned baudRate;  // get set
```

Description

Baud rate override.

---

dataBits

```typescript
TypeScript
dataBits: number;
```

```cs
C#
uint dataBits { get; set; }
```

```cpp
C++
unsigned dataBits;  // get set
```

Description

Data bits override.

---

parity

```typescript
TypeScript
parity: PortParity;
```

```cs
C#
PortParity parity { get; set; }
```

```cpp
C++
PortParity parity;  // get set
```

Description

Parity override.

---

stopBits

```typescript
TypeScript
stopBits: PortStopBits;
```

```cs
C#
PortStopBits stopBits { get; set; }
```

```cpp
C++
PortStopBits stopBits;  // get set
```

Description

Stop bits override.

---

flowControl

```typescript
flowControl: PortFlowControl;
```

```csharp
PortFlowControl flowControl { get; set; }
```

```cpp
PortFlowControl flowControl;  // get set
```

Description

Flow control override.

---

timeouts

```typescript
timeouts: ITimeouts;
```

```csharp
ITimeouts timeouts { get; set; }
```

```cpp
ITimeoutsPtr timeouts;  // get set
```

Description

Port timeouts override.

IDevice PortParity PortStopBits PortFlowControl ITimeouts

IAliasPortDevice Interface

Description

This interface is implemented by an alias port device.

Declaration

```typescript
interface IAliasPortDevice extends IConfigurableDevice {
    // Properties
    aliasPort: number;
    targetDevicePath: string;
}
```

```csharp
public interface IAliasPortDevice : IConfigurableDevice
{
    // Properties
    uint aliasPort { get; set; }
    string targetDevicePath { get; set; }
}
```

```C++
struct IAliasPortDevice : IConfigurableDevice
{
    // Properties
    unsigned aliasPort;  // get set
    _bstr_t targetDevicePath;  // get set
};
```

IAliasPortDevice Properties

---

aliasPort

```TypeScript
aliasPort: number;
```

```C#
uint aliasPort { get; set; }
```

```C++
unsigned aliasPort;  // get set
```

Description

Port number of original serial port this device is an alias for.

---

targetDevicePath

```TypeScript
targetDevicePath: string;
```

```C#
string targetDevicePath { get; set; }
```

```C++
_bstr_t targetDevicePath;  // get set
```

Description

An alternative way to specify the original serial port by its device path.

IBridgePortDevice Interface

Description

This interface is implemented by a local bridge device.

Declaration

TypeScript

```typescript
interface IBridgePortDevice extends IDevice {
    // Properties
    bridgePort: number;
    bridgeServer: string;
    readonly isLocal: boolean;
    readonly isListening: boolean;
    remoteLogin: string;
    remoteDomain: string;
    remotePassword: string;
    readonly securityDescriptor: string;
    emulateBaudrate: boolean;
    emulateTxOverflow: boolean;
    crossoverProbability: number;
    DTR: DestinationPins;
    DSR: DestinationPins;
    DCD: DestinationPins;
    RTS: DestinationPins;
    CTS: DestinationPins;
    RI: DestinationPins;

    // Methods
    restoreDefaultPins(): void;
    startListening(securityDescriptor?: string): void;
}
```

C#

```csharp
public interface IBridgePortDevice : IDevice
{
    // Properties
    uint bridgePort { get; set; }
    string bridgeServer { get; set; }
    bool isLocal { get; }
    bool isListening { get; }
    string remoteLogin { get; set; }
    string remoteDomain { get; set; }
    string remotePassword { get; set; }
    string securityDescriptor { get; }
    bool emulateBaudrate { get; set; }
    bool emulateTxOverflow { get; set; }
    double crossoverProbability { get; set; }
    DestinationPins DTR { get; set; }
    DestinationPins DSR { get; set; }
    DestinationPins DCD { get; set; }
    DestinationPins RTS { get; set; }
    DestinationPins CTS { get; set; }
    DestinationPins RI { get; set; }

    // Methods
    void restoreDefaultPins();
    void startListening(string securityDescriptor);
}
```

```C++
struct IBridgePortDevice : IDevice
{
    // Properties
    unsigned bridgePort;  // get set
    _bstr_t bridgeServer;  // get set
    VARIANT_BOOL isLocal;  // get
    VARIANT_BOOL isListening;  // get
    _bstr_t remoteLogin;  // get set
    _bstr_t remoteDomain;  // get set
    _bstr_t remotePassword;  // get set
    _bstr_t securityDescriptor;  // get
    VARIANT_BOOL emulateBaudrate;  // get set
    VARIANT_BOOL emulateTxOverflow;  // get set
    double crossoverProbability;  // get set
    DestinationPins DTR;  // get set
    DestinationPins DSR;  // get set
    DestinationPins DCD;  // get set
    DestinationPins RTS;  // get set
    DestinationPins CTS;  // get set
    DestinationPins RI;  // get set

    // Methods
    HRESULT restoreDefaultPins();
    HRESULT startListening(_bstr_t securityDescriptor);
};
```

IBridgePortDevice Properties

bridgePort

```TypeScript
bridgePort: number;
```

```C#
uint bridgePort { get; set; }
```

```C++
unsigned bridgePort;  // get set
```

Description

Port number of other side of the local or remote bridge.

bridgeServer

```TypeScript
bridgeServer: string;
```

```C#
string bridgeServer { get; set; }
```

```C++
_bstr_t bridgeServer;  // get set
```

Description

Name or address of the remote server on which a listening port is created. Do not set this property to create a local bridge.

isLocal

```
TypeScript
readonly isLocal: boolean;
```

```
C#
bool isLocal { get; }
```

```
C++
VARIANT_BOOL isLocal;  // get
```

Description

True if this device is part of a local bridge or false otherwise. If `isListening` property is `true`, this property would be `false`.

isListening

```
TypeScript
readonly isListening: boolean;
```

```
C#
bool isListening { get; }
```

```
C++
VARIANT_BOOL isListening;  // get
```

Description

True if this device is in a listening state (that is, “server” side of a remote bridge).

remoteLogin

```
TypeScript
remoteLogin: string;
```

```
C#
string remoteLogin { get; set; }
```

```
C++
_bstr_t remoteLogin;  // get set
```

Description

Optional user name or login to use when connecting to a remote listening port (do not set this property for local bridges).

remoteDomain

```
TypeScript
remoteDomain: string;
```

```C#
string remoteDomain { get; set; }
```

```C++
_bstr_t remoteDomain;  // get set
```

Description

Optional domain name to use when connecting to a remote listening port (do not set this property for local bridges). Can be empty to use default remote computer domain.

---

remotePassword

```TypeScript
remotePassword: string;
```

```C#
string remotePassword { get; set; }
```

```C++
_bstr_t remotePassword;  // get set
```

Description

Optional user password to use when connecting to a remote listening port (do not set this property for local bridges).

This property is write-only.

---

securityDescriptor

```TypeScript
readonly securityDescriptor: string;
```

```C#
string securityDescriptor { get; }
```

```C++
_bstr_t securityDescriptor;  // get
```

Description

This read-only property returns a copy of listening's port security descriptor, in string SD format.

---

emulateBaudrate

```TypeScript
emulateBaudrate: boolean;
```

```C#
bool emulateBaudrate { get; set; }
```

```
C++
VARIANT_BOOL emulateBaudrate;  // get set
```

Description

Turn emulation of baud rate on or off.

By default, Virtual Serial Port Tools does not emulate baud rate.

---

emulateTxOverflow

```
TypeScript
emulateTxOverflow: boolean;
```

```
C#
bool emulateTxOverflow { get; set; }
```

```
C++
VARIANT_BOOL emulateTxOverflow;  // get set
```

Description

Turn emulation of output buffer overflow on or off.

By default, Virtual Serial Port Tools does not emulate output buffer overflow.

---

crossoverProbability

```
TypeScript
crossoverProbability: number;
```

```
C#
double crossoverProbability { get; set; }
```

```
C++
double crossoverProbability;  // get set
```

Description

Single bit crossover probability. May be a floating-point number between 0 and 0.5 (inclusive). Value 0 disables line noise emulation.

By default, Virtual Serial Port Tools does not emulate line noise. Change the value of this property to enable line noise emulation.

DTR

```
TypeScript
DTR: DestinationPins;
```

```
C#
DestinationPins DTR { get; set; }
```

```
C++
DestinationPins DTR;  // get set
```

Description

Allows configuration of custom pin-out for a local DTR line.

DSR

```
TypeScript
DSR: DestinationPins;
```

```
C#
DestinationPins DSR { get; set; }
```

```
C++
DestinationPins DSR;  // get set
```

Description

Allows configuration of custom pin-out for a local DSR line.

DCD

```
TypeScript
DCD: DestinationPins;
```

```
C#
DestinationPins DCD { get; set; }
```

```
C++
DestinationPins DCD;  // get set
```

Description

Allows configuration of custom pin-out for a local DCD line.

RTS

```
TypeScript
RTS: DestinationPins;
```

```
C#
DestinationPins RTS { get; set; }
```

```
C++
DestinationPins RTS;  // get set
```

Description

Allows configuration of custom pin-out for a local RTS line.

CTS

```
TypeScript
CTS: DestinationPins;
```

```
C#
DestinationPins CTS { get; set; }
```

```
C++
DestinationPins CTS;  // get set
```

Description

Allows configuration of custom pin-out for a local CTS line.

RI

```
TypeScript
RI: DestinationPins;
```

```
C#
DestinationPins RI { get; set; }
```

```
C++
DestinationPins RI;  // get set
```

Description

Allows configuration of custom pin-out for a local RI line.

IBridgePortDevice Methods

---

restoreDefaultPins

```
TypeScript
restoreDefaultPins(): void;
```

```
C#
void restoreDefaultPins();
```

```
C++
HRESULT restoreDefaultPins();
```

Description

This method restores default pin-out assignment.

---

startListening

```
TypeScript
startListening(securityDescriptor?: string): void;
```

```
C#
void startListening(string securityDescriptor);
```

```
C++
HRESULT startListening(_bstr_t securityDescriptor);
```

Parameters

`securityDescriptor`

Optional listening port's security descriptor, in string SD format. If omitted, or empty string is passed, defaults to allowing “Everyone” to connect to the port.

Description

Starts listening port, optionally applying passed security descriptor. After this method returns, a remote bridge device may connect to this listening port.

Note that the actual connection occurs only when the client bridge port is opened by some application.

IDevice

ISharedPortDevice Interface

Description

This interface is implemented by a shared port device.

Declaration

```
TypeScript
interface ISharedPortDevice extends IConfigurableDevice {
    // Properties
    sharedPort: number;
}
```

```
C#
public interface ISharedPortDevice : IConfigurableDevice
{
    // Properties
    uint sharedPort { get; set; }
}
```

```
C++
struct ISharedPortDevice : IConfigurableDevice
{
    // Properties
    unsigned sharedPort;  // get set
};
```

ISharedPortDevice Properties

sharedPort

```
TypeScript
sharedPort: number;
```

```C#
uint sharedPort { get; set; }
```

```C++
unsigned sharedPort;  // get set
```

Description

Port number of original serial port this device is sharing.

IConfigurableDevice

ITcpPortDevice Interface

Description

This interface is implemented by a tcp port device.

Declaration

```TypeScript
interface ITcpPortDevice extends IConfigurableDevice {
    // Properties
    remoteHost: string;
    remoteTcpPort: number;
    localAddress: string;
    localTcpPort: number;
    protocol: TcpPortProtocol;
    reconnectTimeout: number;
    bufferSize: number;
}
```

```C#
public interface ITcpPortDevice : IConfigurableDevice
{
    // Properties
    string remoteHost { get; set; }
    uint remoteTcpPort { get; set; }
    string localAddress { get; set; }
    uint localTcpPort { get; set; }
    TcpPortProtocol protocol { get; set; }
    uint reconnectTimeout { get; set; }
    uint bufferSize { get; set; }
}
```

```C++
struct ITcpPortDevice : IConfigurableDevice
{
    // Properties
    _bstr_t remoteHost;  // get set
    unsigned remoteTcpPort;  // get set
    _bstr_t localAddress;  // get set
    unsigned localTcpPort;  // get set
    TcpPortProtocol protocol;  // get set
    unsigned reconnectTimeout;  // get set
    unsigned bufferSize;  // get set
};
```

ITcpPortDevice Properties

remoteHost

```typescript
TypeScript
remoteHost: string;
```

```csharp
C#
string remoteHost { get; set; }
```

```cpp
C++
_bstr_t remoteHost;  // get set
```

Description

Remote endpoint's host name or address. DNS, NETBIOS and other types of host names are supported as well as IPv4 and IPv6 IP addresses.

Setting this property sets the virtual serial port to “connecting” mode. If the property is read and the virtual port is in “listening” mode, an empty string is returned.

---

remoteTcpPort

```typescript
TypeScript
remoteTcpPort: number;
```

```csharp
C#
uint remoteTcpPort { get; set; }
```

```cpp
C++
unsigned remoteTcpPort;  // get set
```

Description

Remote endpoint's TCP port number.

Setting this property sets the virtual serial port to “connecting” mode. If the property is read and the virtual port is in “listening” mode, a value of zero is returned.

---

localAddress

```typescript
TypeScript
localAddress: string;
```

```csharp
C#
string localAddress { get; set; }
```

```cpp
C++
_bstr_t localAddress;  // get set
```

Description

Local listening IPv4 or IPv6 address. Can also be `*` to specify that the port should listen on all local addresses.

Setting this property sets the virtual serial port to “listening” mode. If the property is read and the virtual port is in “connecting” mode, an empty string is returned.

---

localTcpPort

```
TypeScript
localTcpPort: number;
```

```
C#
uint localTcpPort { get; set; }
```

```
C++
unsigned localTcpPort;  // get set
```

Description

Local listening TCP port number.

Setting this property sets the virtual serial port to "listening" mode. If the property is read and the virtual port is in "connecting" mode, a value of zero is returned.

---

protocol

```
TypeScript
protocol: TcpPortProtocol;
```

```
C#
TcpPortProtocol protocol { get; set; }
```

```
C++
TcpPortProtocol protocol;  // get set
```

Description

Set or retrieve the virtual serial port protocol.

---

reconnectTimeout

```
TypeScript
reconnectTimeout: number;
```

```
C#
uint reconnectTimeout { get; set; }
```

```
C++
unsigned reconnectTimeout;  // get set
```

Description

The value of reconnection timeout, in milliseconds. This is the time that virtual serial port waits until trying to re-establish a broken network connection. Only relevant to virtual serial ports in "connecting" mode.

---

bufferSize

```typescript
bufferSize: number;
```

```csharp
uint bufferSize { get; set; }
```

```cpp
unsigned bufferSize;  // get set
```

Description

Internal TCP buffer size, in bytes. The default value is 512 KB. Supported range is 4 KB..10 MB. Prefer larger values for high-speed transfers.

IConfigurableDevice

IRemotePortDevice Interface

Description

This interface is implemented by a remote port device.

Declaration

```typescript
interface IRemotePortDevice extends IDevice {
    // Properties
    remoteHost: string;
    remotePort: number;
    connectionTimeout: number;
    connectionAttempts: number;
    login: string;
    password: string;
    domain: string;
}
```

```csharp
public interface IRemotePortDevice : IDevice
{
    // Properties
    string remoteHost { get; set; }
    uint remotePort { get; set; }
    uint connectionTimeout { get; set; }
    uint connectionAttempts { get; set; }
    string login { get; set; }
    string password { set; }
    string domain { get; set; }
}
```

```cpp
struct IRemotePortDevice : IDevice
{
    // Properties
    _bstr_t remoteHost;  // get set
    unsigned int remotePort;  // get set
    unsigned int connectionTimeout;  // get set
    unsigned int connectionAttempts;  // get set
    _bstr_t login;  // get set
    _bstr_t password;  // set
    _bstr_t domain;  // get set
};
```

IRemotePortDevice Properties

remoteHost

```TypeScript
remoteHost: string;
```

```C#
string remoteHost { get; set; }
```

```C++
_bstr_t remoteHost;  // get set
```

Description

Name or address of a remote host this port is associated with.

---

remotePort

```TypeScript
remotePort: number;
```

```C#
uint remotePort { get; set; }
```

```C++
unsigned int remotePort;  // get set
```

Description

Port number on a remote server (specified by `remoteHost` property) this port is associated with.

---

connectionTimeout

```TypeScript
connectionTimeout: number;
```

```C#
uint connectionTimeout { get; set; }
```

```C++
unsigned int connectionTimeout;  // get set
```

Description

Connection timeout, in milliseconds. When the local port is opened by application, an attempt to establish connection is made for a given number of milliseconds before returning error code.

---

connectionAttempts

```TypeScript
connectionAttempts: number;
```

```C#
uint connectionAttempts { get; set; }
```

```C++
unsigned int connectionAttempts;  // get set
```

Description

Number of connection attempts before giving up.

---

login

```TypeScript
login: string;
```

```C#
string login { get; set; }
```

```C++
_bstr_t login;  // get set
```

Description

User name for authentication on the remote server.

---

password

```TypeScript
password: string;
```

```C#
string password { set; }
```

```C++
_bstr_t password;  // set
```

Description

User password for authentication on the remote server. For security reasons, this property is write-only.

---

domain

```TypeScript
domain: string;
```

```C#
string domain { get; set; }
```

```C++
_bstr_t domain;  // get set
```

Description

User domain name (optional) for authentication on the remote server.

Example

The following code snippet illustrates the creation and configuration of a remote virtual serial port:

```
TypeScript
// The following function gets the name of the remote server
// and its port number, as well as user credentials,  creates a virtual serial port, connects
// it to the server and returns the port name
function connectPort(remoteHost: string, remotePort: number, login: string,
        password: string, domain: string) : string {
    var library = (IRemotePortLibrary) new ActiveXObject("hhdvspkit.SerialPortLibrary.1");
    var port = library.createRemotePort();

    port.remoteHost = remoteHost;
    port.remotePort = remotePort;
    port.login = login;
    port.password = password;
    port.domain = domain;

    return port.devicePath;
}
```

IDevice

IRemotePortDescription Interface

```
TypeScript
interface IRemotePortDescription {
    // Properties
    readonly name: string;
    readonly port: number;
}
```

```
C#
public interface IRemotePortDescription
{
    // Properties
    string name { get; }
    uint port { get; }
}
```

```
C++
struct IRemotePortDescription : IDispatch
{
    // Properties
    _bstr_t name;  // get
    unsigned int port;  // get
};
```

IRemotePortDescription Properties

name

```
TypeScript
readonly name: string;
```

```
C#
string name { get; }
```

```C++
_bstr_t name;  // get
```

Description

Holds the remote port name. This property is read-only.

---

port

```TypeScript
readonly port: number;
```

```C#
uint port { get; }
```

```C++
unsigned int port;  // get
```

Description

Holds the remote port number. This property is read-only.

IPipePortDevice Interface

Description

This interface is implemented by a pipe port device.

Declaration

```TypeScript
interface IPipePortDevice extends IDevice {
    // Properties
    readonly pipeName: string;
    readonly numberOfInstances: number;
    readonly outputBufferSize: number;
    readonly inputBufferSize: number;
    readonly defaultTimeout: number;
    readonly securityDescriptor: string;

    // Methods
    configureCreatePipe(pipeName: string,
        numberOfInstances: number,
        outputBufferSize: number,
        inputBufferSize: number,
        defaultTimeout: number): void;
    configureCreatePipe2(pipeName: string,
        securityDescriptor: string,
        numberOfInstances: number,
        outputBufferSize: number,
        inputBufferSize: number,
        defaultTimeout: number): void;
    configureConnectPipe(pipeName: string): void;
}
```

```C#
public interface IPipePortDevice : IDevice
{
    // Properties
    string pipeName { get; }
    uint numberOfInstances { get; }
    uint outputBufferSize { get; }
    uint inputBufferSize { get; }
    uint defaultTimeout { get; }
    string securityDescriptor { get; }

    // Methods
    void configureCreatePipe(string pipeName,
        uint numberOfInstances,
        uint outputBufferSize,
        uint inputBufferSize,
        uint defaultTimeout);
    void configureCreatePipe2(string pipeName,
        string securityDescriptor,
        uint numberOfInstances,
        uint outputBufferSize,
        uint inputBufferSize,
        uint defaultTimeout);
    void configureConnectPipe(string pipeName);
}
```

```C++
struct IPipePortDevice : IDevice
{
    // Properties
    _bstr_t pipeName;  // get
    unsigned numberOfInstances;  // get
    unsigned outputBufferSize;  // get
    unsigned inputBufferSize;  // get
    unsigned defaultTimeout;  // get
    _bstr_t securityDescriptor;  // get

    // Methods
    HRESULT configureCreatePipe(_bstr_t pipeName,
        unsigned numberOfInstances,
        unsigned outputBufferSize,
        unsigned inputBufferSize,
        unsigned defaultTimeout);
    HRESULT configureCreatePipe2(_bstr_t pipeName,
        _bstr_t securityDescriptor,
        unsigned numberOfInstances,
        unsigned outputBufferSize,
        unsigned inputBufferSize,
        unsigned defaultTimeout);
    HRESULT configureConnectPipe(_bstr_t pipeName);
};
```

IPipePortDevice Properties

pipeName

```TypeScript
readonly pipeName: string;
```

```C#
string pipeName { get; }
```

```C++
_bstr_t pipeName;  // get
```

Description

The full name of the pipe this port is associated with. The pipe name returned is of the form `\\servername\pipe\pipename`.

---

numberOfInstances

```TypeScript
readonly numberOfInstances: number;
```

```C#
uint numberOfInstances { get; }
```

```C++
unsigned numberOfInstances;  // get
```

Description

The maximum number of instances that can be created for this pipe. The first instance of the pipe can specify this value; the same number must be specified for other instances of the pipe. Acceptable values are in the range 1 through 255. 255 means “unlimited instances”, in this case the number of pipe instances that can be created is limited only by the availability of system resources.

---

outputBufferSize

```TypeScript
readonly outputBufferSize: number;
```

```C#
uint outputBufferSize { get; }
```

```C++
unsigned outputBufferSize;  // get
```

Description

The number of bytes to reserve for the output buffer. This value is only a hint.

---

inputBufferSize

```TypeScript
readonly inputBufferSize: number;
```

```C#
uint inputBufferSize { get; }
```

```C++
unsigned inputBufferSize;  // get
```

Description

The number of bytes to reserve for the input buffer. This value is only a hint.

defaultTimeout

```TypeScript
readonly defaultTimeout: number;
```

```C#
uint defaultTimeout { get; }
```

```C++
unsigned defaultTimeout;  // get
```

Description

The default time-out value, in milliseconds. Each instance of a named pipe must specify the same value. A value of zero will result in a default time-out of 50 milliseconds.

securityDescriptor

```TypeScript
readonly securityDescriptor: string;
```

```C#
string securityDescriptor { get; }
```

```C++
_bstr_t securityDescriptor;  // get
```

Description

Holds the current pipe security descriptor. If empty, default security descriptor grants connection rights to `Everyone`.

IPipePortDevice Methods

configureCreatePipe

```TypeScript
configureCreatePipe(pipeName: string,
    numberOfInstances: number,
    outputBufferSize: number,
    inputBufferSize: number,
    defaultTimeout: number): void;
```

```C#
void configureCreatePipe(string pipeName,
    uint numberOfInstances,
    uint outputBufferSize,
    uint inputBufferSize,
    uint defaultTimeout);
```

```C++
HRESULT configureCreatePipe(_bstr_t pipeName,
    unsigned numberOfInstances,
    unsigned outputBufferSize,
    unsigned inputBufferSize,
    unsigned defaultTimeout);
```

Parameters

`pipeName`

The name of the pipe to create. Pipe name must have the following format: `\\.\pipe\pipename` where `pipename` is a unique pipe name (possibly with path).

`numberOfInstances`

The maximum number of instances that can be created for this pipe. The first instance of the pipe can specify this value; the same number must be specified for other instances of the pipe. Acceptable values are in the range 1 through 255. 255 means “unlimited instances”, in this case the number of pipe instances that can be created is limited only by the availability of system resources.

`outputBufferSize`

The number of bytes to reserve for the output buffer. This value is only a hint.

`inputBufferSize`

The number of bytes to reserve for the input buffer. This value is only a hint.

`defaultTimeout`

The default time-out value, in milliseconds. Each instance of a named pipe must specify the same value. A value of zero will result in a default time-out of 50 milliseconds.

Description

Configures the port to automatically created a given named pipe whenever it is opened by an application. If an error occurs during creation of the pipe, this error is propagated back to the opener of the virtual serial port.

---

configureCreatePipe2

```TypeScript
configureCreatePipe2(pipeName: string,
    securityDescriptor: string,
    numberOfInstances: number,
    outputBufferSize: number,
    inputBufferSize: number,
    defaultTimeout: number): void;
```

```C#
void configureCreatePipe2(string pipeName,
    string securityDescriptor,
    uint numberOfInstances,
    uint outputBufferSize,
    uint inputBufferSize,
    uint defaultTimeout);
```

```C++
HRESULT configureCreatePipe2(_bstr_t pipeName,
    _bstr_t securityDescriptor,
    unsigned numberOfInstances,
    unsigned outputBufferSize,
    unsigned inputBufferSize,
    unsigned defaultTimeout);
```

Parameters

`pipeName`

The name of the pipe to create. Pipe name must have the following format: `\\.\pipe\pipename` where `pipename` is a unique pipe name (possibly with path).

`securityDescriptor`

Pipe security descriptor in SDDL format. Security descriptor must specify `GENERIC_READ`, `GENERIC_WRITE` and `SYNCHRONIZE` standard rights to groups or users. If empty string is specified, by default, pipe access is granted to the `Everyone` group.

`numberOfInstances`

The maximum number of instances that can be created for this pipe. The first instance of the pipe can specify this value; the same number must be specified for other instances of the pipe. Acceptable values are in the range 1 through 255. 255 means “unlimited instances”, in this case the number of pipe instances that can be created is limited only by the availability of system resources.

`outputBufferSize`

The number of bytes to reserve for the output buffer. This value is only a hint.

`inputBufferSize`

The number of bytes to reserve for the input buffer. This value is only a hint.

`defaultTimeout`

The default time-out value, in milliseconds. Each instance of a named pipe must specify the same value. A value of zero will result in a default time-out of 50 milliseconds.

Description

Configures the port to automatically created a given named pipe whenever it is opened by an application. If an error occurs during creation of the pipe, this error is propagated back to the opener of the virtual serial port.

---

configureConnectPipe

```TypeScript
configureConnectPipe(pipeName: string): void;
```

```C#
void configureConnectPipe(string pipeName);
```

```C++
HRESULT configureConnectPipe(_bstr_t pipeName);
```

Parameters

`pipeName`

The full name of the pipe. It must be a string of the following format: `\\servername\pipe\pipename`,

where `servername` is pipe's server name, DNS name, IP address or any other supported locator and `pipename` is a unique pipe name (possibly with a path).

Description

Configures the port to automatically connect to a given pipe whenever it is opened by an application. If an error occurs during connection to the pipe, this error is propagated back to the opener of the virtual serial port.

IDevice

IScriptPortDevice Interface

Description

This interface is implemented by a script port device.

Declaration

TypeScript

```typescript
interface IScriptPortDevice extends IConfigurableDevice {
    // Properties
    readonly scriptPath: string;
    readonly validationErrors: string;
    logPath: string;
    initializationValue: string;

    // Methods
    setScriptFile(path: string): boolean;
    setScriptText(scriptBody: string): boolean;
    setScriptParam(name: string, value: string): void;
}
```

C#

```csharp
public interface IScriptPortDevice : IConfigurableDevice
{
    // Properties
    string scriptPath { get; }
    string validationErrors { get; }
    string logPath { get; set; }
    string initializationValue { get; set; }

    // Methods
    bool setScriptFile(string path);
    bool setScriptText(string scriptBody);
    void setScriptParam(string name, string value);
}
```

C++

```cpp
struct IScriptPortDevice : IConfigurableDevice
{
    // Properties
    _bstr_t scriptPath;  // get
    _bstr_t validationErrors;  // get
    _bstr_t logPath;  // get set
    _bstr_t initializationValue;  // get set

    // Methods
    VARIANT_BOOL setScriptFile(_bstr_t path);
    VARIANT_BOOL setScriptText(_bstr_t scriptBody);
    HRESULT setScriptParam(_bstr_t name, _bstr_t value);
};
```

IScriptPortDevice Properties

scriptPath

```
TypeScript
readonly scriptPath: string;
```

```
C#
string scriptPath { get; }
```

```
C++
_bstr_t scriptPath;  // get
```

Description

A full path to a script file. This property is read-only. To change the port's script, use the `setScriptFile` method. If a script has been set with a call to `setScriptText` method, this property returns an empty string.

---

validationErrors

```
TypeScript
readonly validationErrors: string;
```

```
C#
string validationErrors { get; }
```

```
C++
_bstr_t validationErrors;  // get
```

Description

This property stores the script validation error, if any. You can query this property after `setScriptFile` method returns `false`.

---

logPath

```
TypeScript
logPath: string;
```

```
C#
string logPath { get; set; }
```

```
C++
_bstr_t logPath;  // get set
```

Description

A full path of a folder used to store script execution logs. By default is set to `%TEMP%\vspt_script_logs`.

---

initializationValue

```
TypeScript
initializationValue: string;
```

```C#
string initializationValue { get; set; }
```

```C++
_bstr_t initializationValue;  // get set
```

Description

A script initialization parameter. Can be any string. Note that the length of this string is limited to 5,000 characters. An attempt to set longer value will result in exception.

IScriptPortDevice Methods

setScriptFile

```TypeScript
setScriptFile(path: string): boolean;
```

```C#
bool setScriptFile(string path);
```

```C++
VARIANT_BOOL setScriptFile(_bstr_t path);
```

Parameters

`path`

The full path to a script file.

Description

Updates the virtual port's script. Library compiles and validates the script and stores its contents internally. The script port will not read the original file after this method returns successfully. This is done for security reasons: an attempt to modify the script after the port has been created will not lead to undesired updated behavior.

The read-only `scriptPath` property will hold the path to the original script file for reference.

The method returns `true` on success and `false` if script compilation and validation fails. In the latter case, more information is available by querying the `validationErrors` property. This method may also throw an exception if script file does not contain required elements.

setScriptText

```TypeScript
setScriptText(scriptBody: string): boolean;
```

```C#
bool setScriptText(string scriptBody);
```

```C++
VARIANT_BOOL setScriptText(_bstr_t scriptBody);
```

Parameters

`scriptBody`

The device script contents.

Description

Updates the virtual port's script. Library compiles and validates the script and stores its contents internally.

The read-only `scriptPath` property will be set to an empty string after calling this method.

The method returns `true` on success and `false` if script compilation and validation fails. In the latter case, more information is available by querying the `validationErrors` property. This method may also throw an exception if script file does not contain required elements.

---

setScriptParam

```
TypeScript
setScriptParam(name: string, value: string): void;
```

```
C#
void setScriptParam(string name, string value);
```

```
C++
HRESULT setScriptParam(_bstr_t name, _bstr_t value);
```

Parameters

`name`

Custom parameter name.

`value`

Custom parameter value.

Description

This method passes a custom parameter (specified by its `name` and `value`) to a port's script. The port must be opened by some application in order for this method to succeed. If the port is not opened, an exception is thrown. Otherwise, script's IScriptDevice.setParam method is called. If this method is not defined or throws an exception, `setScriptParam` method also throws an exception.

It is up to the port's script to interpret the meaning of custom parameter's name and value.

There is an alternative way of setting a custom script parameter: an application that opens a virtual script port may send a custom IOCTL_SCRIPTPORT_SET_PARAM device I/O control request.

IConfigurableDevice

ITimeouts Interface

Description

This interface is implemented by a timeouts override object. Values stored in this object are documented in the Windows Documentation. IConfigurableDevice.timeouts property is used to set or retrieve this object. A new object with specified values may also be created using the

ISerialPortLibrary.createTimeoutsObject method.

Declaration

```typescript
interface ITimeouts {
    // Properties
    readIntervalTimeout: number;
    readTotalTimeoutMultiplier: number;
    readTotalTimeoutConstant: number;
    writeTotalTimeoutMultiplier: number;
    writeTotalTimeoutConstant: number;
}
```

```csharp
public interface ITimeouts
{
    // Properties
    uint readIntervalTimeout { get; set; }
    uint readTotalTimeoutMultiplier { get; set; }
    uint readTotalTimeoutConstant { get; set; }
    uint writeTotalTimeoutMultiplier { get; set; }
    uint writeTotalTimeoutConstant { get; set; }
}
```

```cpp
struct ITimeouts : IDispatch
{
    // Properties
    unsigned readIntervalTimeout;  // get set
    unsigned readTotalTimeoutMultiplier;  // get set
    unsigned readTotalTimeoutConstant;  // get set
    unsigned writeTotalTimeoutMultiplier;  // get set
    unsigned writeTotalTimeoutConstant;  // get set
};
```

ITimeouts Properties

readIntervalTimeout

```typescript
readIntervalTimeout: number;
```

```csharp
uint readIntervalTimeout { get; set; }
```

```cpp
unsigned readIntervalTimeout;  // get set
```

Description

The maximum time allowed to elapse before the arrival of the next byte on the communications line, in milliseconds. If the interval between the arrival of any two bytes exceeds this amount, the ReadFile operation is completed and any buffered data is returned. A value of zero indicates that interval time-outs are not used.

A value of `4294967295` (or `-1`), combined with zero values for both the `readTotalTimeoutConstant` and `readTotalTimeoutMultiplier` members, specifies that the read operation is to return immediately with the bytes that have already been received, even if no bytes have been received.

readTotalTimeoutMultiplier

```
TypeScript
readTotalTimeoutMultiplier: number;
```

```
C#
uint readTotalTimeoutMultiplier { get; set; }
```

```
C++
unsigned readTotalTimeoutMultiplier;  // get set
```

Description

The multiplier used to calculate the total time-out period for read operations, in milliseconds. For each read operation, this value is multiplied by the requested number of bytes to be read.

---

readTotalTimeoutConstant

```
TypeScript
readTotalTimeoutConstant: number;
```

```
C#
uint readTotalTimeoutConstant { get; set; }
```

```
C++
unsigned readTotalTimeoutConstant;  // get set
```

Description

A constant used to calculate the total time-out period for read operations, in milliseconds. For each read operation, this value is added to the product of the `readTotalTimeoutMultiplier` member and the requested number of bytes.

A value of zero for both the `readTotalTimeoutMultiplier` and `readTotalTimeoutConstant` members indicates that total time-outs are not used for read operations.

---

writeTotalTimeoutMultiplier

```
TypeScript
writeTotalTimeoutMultiplier: number;
```

```
C#
uint writeTotalTimeoutMultiplier { get; set; }
```

```
C++
unsigned writeTotalTimeoutMultiplier;  // get set
```

Description

The multiplier used to calculate the total time-out period for write operations, in milliseconds. For each write operation, this value is multiplied by the number of bytes to be written.

writeTotalTimeoutConstant

```typescript
writeTotalTimeoutConstant: number;
```

```csharp
uint writeTotalTimeoutConstant { get; set; }
```

```cpp
unsigned writeTotalTimeoutConstant;  // get set
```

Description

A constant used to calculate the total time-out period for write operations, in milliseconds. For each write operation, this value is added to the product of the `writeTotalTimeoutMultiplier` member and the number of bytes to be written.

A value of zero for both the `writeTotalTimeoutMultiplier` and `writeTotalTimeoutConstant` members indicates that total time-outs are not used for write operations.

IOpeningInfo Interface

Description

An object implementing this interface is obtained through the IDevice.openingInfo property. This object properties can be queried to get information about the process that opened the virtual serial port and port opening parameters.

Declaration

```typescript
interface IOpeningInfo {
    // Properties
    readonly baudRate: number;
    readonly byteSize: number;
    readonly parity: PortParity;
    readonly stopBits: PortStopBits;
    readonly processId: number;
    readonly processName: string;
}
```

```csharp
public interface IOpeningInfo
{
    // Properties
    uint baudRate { get; }
    uint byteSize { get; }
    PortParity parity { get; }
    PortStopBits stopBits { get; }
    uint processId { get; }
    string processName { get; }
}
```

```C++
struct IOpeningInfo : IDispatch
{
    // Properties
    unsigned baudRate;  // get
    unsigned byteSize;  // get
    PortParity parity;  // get
    PortStopBits stopBits;  // get
    unsigned processId;  // get
    _bstr_t processName;  // get
};
```

IOpeningInfo Properties

baudRate

```TypeScript
readonly baudRate: number;
```

```C#
uint baudRate { get; }
```

```C++
unsigned baudRate;  // get
```

Description

Baud rate, in bits per second.

byteSize

```TypeScript
readonly byteSize: number;
```

```C#
uint byteSize { get; }
```

```C++
unsigned byteSize;  // get
```

Description

Byte size, in bits.

parity

```TypeScript
readonly parity: PortParity;
```

```C#
PortParity parity { get; }
```

```C++
PortParity parity;  // get
```

Description

Parity value.

---

stopBits

```TypeScript
readonly stopBits: PortStopBits;
```

```C#
PortStopBits stopBits { get; }
```

```C++
PortStopBits stopBits;  // get
```

Description

Stop bits.

---

processId

```TypeScript
readonly processId: number;
```

```C#
uint processId { get; }
```

```C++
unsigned processId;  // get
```

Description

ID of the process that opened the port.

---

processName

```TypeScript
readonly processName: string;
```

```C#
string processName { get; }
```

```C++
_bstr_t processName;  // get
```

Description

Full file name path of the process that opened the port.

PortParity PortStopBits

SerialPortType Enumeration

Symbol	Value	Description
`Remote`	`0`	Specifies remote port device.
`Shared`	`1`	Specifies shared port device.
`Alias`	`2`	Specifies alias port device.
`Bridge`	`3`	Specifies bridge port device.
`Pipe`	`4`	Specifies pipe port device.
`Tcp`	`5`	Specifies TCP/IP port device.
`Script`	`6`	Specifies Script port device.

PortParity Enumeration

Description

Supported parity constants.

Symbol	Value	Description
`NoParity`	`0`	No parity.
`OddParity`	`1`	Odd parity.
`EvenParity`	`2`	Even parity.
`MarkParity`	`3`	Mark parity.
`SpaceParity`	`4`	Space parity.

PortStopBits Enumeration

Description

Supported stop bits constants.

Symbol	Value	Description
`One`	`0`	1 stop bit.
`OneAndHalf`	`1`	1.5 stop bits.
`Two`	`2`	2 stop bits.

PortFlowControl Enumeration

Description

Supported flow control constants.

Symbol	Value	Description
`NotSet`	`0`	Specific flow control is not enforced.
`None`	`1`	None flow control is enforced.
`Software`	`2`	Software (XON/XOFF) flow control is enforced.
`Hardware`	`3`	Hardware flow control is enforced.

DestinationPins Enumeration

Description

Destination pins for custom pin-out. See the Bridge Pin-Outs section for additional information.

Symbol	Value	Description
`NotConnected`	`0`	The given pin is not connected to any other pins.
`Local_DTR`	`0x1`	The given pin is connected to local DTR pin.
`Local_DTS`	`0x2`	The given pin is connected to local DTS pin.
`Local_DCD`	`0x4`	The given pin is connected to local DCD pin.
`Local_RTS`	`0x8`	The given pin is connected to local RTS pin.
`Local_CTS`	`0x10`	The given pin is connected to local CTS pin.
`Local_RI`	`0x20`	The given pin is connected to local RI pin.
`Remote_DTR`	`0x100`	The given pin is connected to remote DTR pin.
`Remote_DTS`	`0x200`	The given pin is connected to remote DTS pin.
`Remote_DCD`	`0x400`	The given pin is connected to remote DCD pin.
`Remote_RTS`	`0x800`	The given pin is connected to remote RTS pin.
`Remote_CTS`	`0x1000`	The given pin is connected to remote CTS pin.
`Remote_RI`	`0x2000`	The given pin is connected to remote RI pin.

CompatibilityFlags Enumeration

Description

A set of compatibility flags that change the behavior of virtual serial ports to increase compatibility with legacy software.

Symbol	Value	Description
`IgnoreInvalidSpecialChars`	`1`	Virtual serial ports must silently ignore invalid special characters set by application. By default, when this compatibility flag is not set, virtual serial ports fail requests that attempt to set invalid combination of special characters, as required by specification.
`IgnoreTxClearPurgeFlag`	`2`	Virtual serial devices will ignore serial purge requests with TXCLEAR flag. This prevents unwanted behavior if the request is erroneously sent by application while there is still data in transmit queue. Currently applies to local bridges only.

TcpPortProtocol Enumeration

Description

Specify the protocol for TCP/IP Port. See the ITcpPortDevice.protocol property for more information.

Symbol	Value	Description
`Rfc2217`	`0`	The protocol corresponds to RFC2217 standard.
`Raw`	`0x1`	The protocol is raw.

Device Script API

This section provides detailed API documentation for a device script, a script that powers up a script port virtual serial device.

Device Script API is fully asynchronous, most methods and some properties return promise objects. Promises are resolved when appropriate events happen, make sure your code properly work with asynchronous calls. It is recommended to mark your functions that call those methods with `async` keyword and use the `await` operator. A callback model is also supported, you can query continuations using the `Promise.then` method.

> **NOTE**
>
> This script *execution context* is created after the port is opened and is destroyed as soon as it is closed. Any global variables you create will only live this long.
>
> That is, you cannot store any state between port opening attempts. However, the execution context is guaranteed to exist until the port is closed by the application. Anyway, prefer storing any state in the created device script object, not on a global scope.

Global Object

Methods and properties defined on a Global object are described in the IGlobals topic.

- port property returns a reference to a global Port API IPort object. It is used by a device script to communicate with a virtual serial port device driver.
- fs property returns a reference to a global IFileManager object. Device script uses this object to access local file system.
- net property returns a reference to a global INetworkManager object. Device script uses this object to create TCP and UDP sockets and communicate over the network.
- http property returns a reference to a global IHttpClient object. Device script uses this object to make HTTP requests.
- log method is used to perform a diagnostic output.
- delay method produces a promise that is automatically resolved after a given amount of time.
- async and cancelAsync methods allow device script to schedule delayed and optionally repetitive continuations.

File System Object

Global file system object (implemented as IFileManager interface) provides convenient access to the local file system, allowing the device script to read and store data in files, create or delete files and folders as well as enumerate the contents of a folder.

- Create and delete folders using the IFileManager.createFolder and IFileManager.deleteFolder methods.
- Create or open files using the IFileManager.createFile method.
- Delete files with IFileManager.deleteFile method.
- Copy or move(rename) files using the IFileManager.copyFile and IFileManager.moveFile methods.
- Enumerate contents of a folder using the IFileManager.enumFiles method.
- Load a text file with IFileManager.loadTextFile method.
- Read/write data from/to a file using the IFile.read and IFile.write methods.

Network Object

The network object (implemented as INetworkManager interface) provides a way to create TCP and UDP sockets, connect or bind them and perform a data transfer over the network.

- Create TCP and UDP sockets using the INetworkManager.createTcpSocket and INetworkManager.createUdpSocket methods.
- Create TCP listener object using the INetworkManager.createTcpListener method.
- Connect a socket using the ISocket.connect method, or bind it using the IUdpSocket.bind method.
- Perform a data transfer using the ISocket.send and ISocket.receive methods.

HTTP Object

The Http object (implemented as IHttpClient interface) provides the ability for a device script to make HTTP requests to query and control remote HTTP resources.

- Build and send an HTTP request using the IHttpClient.request method.
- Alternatively, use the "shortcut" methods, such as IHttpClient.get, IHttpClient.post, IHttpClient.getString or IHttpClient.getBlob.

Global Interface

Description

This is a "virtual" interface. All its properties and methods are actually available on a Global JavaScript object.

Declaration

```TypeScript
interface IGlobals {
    // Properties
    readonly port: Port.IPort;
    readonly fs: FS.IFileManager;
    readonly net: Net.INetworkManager;
    readonly http: Http.IHttpClient;

    // Methods
    log(value: any): void;
    delay(ms: number): Promise<void>;
    async(handler: () => void, ms: number, repetitive?: boolean): number;
    cancelAsync(handlerId: number): void;
    createDevice(initializationValue?: string): Port.IScriptDevice;
    createDeviceAsync(initializationValue?: string): Promise<Port.IScriptDevice>;
}
```

```C#
// This interface is not available in managed environment
```

```C++
// This interface is not available in native environment
```

IGlobals Properties

port

```TypeScript
readonly port: Port.IPort;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

A reference to a global Port API object. Use this object to communicate with a virtual script serial port device driver.

---

fs

```TypeScript
readonly fs: FS.IFileManager;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

A reference to a global `FS.IFileManager` object. Device script uses this object to access local file system.

---

net

```TypeScript
readonly net: Net.INetworkManager;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

A reference to a global `Net.INetworkManager` object. Device script uses this object to create TCP and UDP sockets and use them to communicate over the network.

---

http

```TypeScript
readonly http: Http.IHttpClient;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

A reference to a global `Http.IHttpClient` object. Device script uses this object to make HTTP requests.

IGlobals Methods

log

```TypeScript
log(value: any): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`value`

An object to dump to a log file. A passed object is first converted to a string, it is not a string already.

Description

Dump a given value to a string.

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`:

```
async function test() {
    // ...
    await delay(500);
    // ...
}
```

Using continuations:

```
delay(500).then(() => { ... });
```

---

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 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.

---

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.

---

createDevice

```TypeScript
createDevice(initializationValue?: string): Port.IScriptDevice;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`initializationValue`

An optional initialization value, set using a IScriptPortDevice.initializationValue property.

Return Value

A reference to a created port device object.

Description

This method must be implemented by a device script. It is invoked each time the port is opened by application. It can perform an optional initialization and must create an instance of a class that derives the Port.IScriptDevice interface and return it.

Either this global function or createDeviceAsync must be defined, otherwise the script will fail validation.

Define this function if you don't require asynchronous initialization.

---

createDeviceAsync

```TypeScript
createDeviceAsync(initializationValue?: string): Promise<Port.IScriptDevice>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`initializationValue`

An optional initialization value, set using a IScriptPortDevice.initializationValue property.

Return Value

A reference to a created port device object.

Description

This method must be implemented by a device script. It is invoked each time the port is opened by application. It can perform an optional initialization and must create an instance of a class that derives the Port.IScriptDevice interface and return it.

Either this global function or createDevice must be defined, otherwise the script will fail validation.

Define this function if you require asynchronous initialization.

Port API

IPort Interface

Description

This interface is implemented by virtual serial port driver and represents a virtual serial port device opened by application. It is available as a global port object.

Declaration

TypeScript
```
interface IPort {

    // Methods
    provideReceivedData(text: string, encoding?: Common.Encoding | number): void;
    provideReceivedData(byte: number): void;
    provideReceivedData(bytes: number[] | Uint8Array | Uint16Array | Uint32Array | ArrayBuffer |
DataView): void;
    provideReceivedData(sendAs: Port.WriteAs,
        data: number[] | Uint8Array | Uint16Array | Uint32Array | ArrayBuffer | DataView,
        bigEndian?: boolean): void;
    getSentData(maxBytes?: number): Promise<Uint8Array>;
    setError(errorCode: number): void;
    setEventMask(mask: Port.EventMask): void;
    clearEventMask(): void;
}
```

C#
```
// This interface is not available in managed environment
```

C++
```
// This interface is not available in native environment
```

IPort Methods

---

provideReceivedData

TypeScript
```
provideReceivedData(text: string, encoding?: Common.Encoding | number): void;
```

C#
```
// This method is not available in managed environment
```

C++
```
// This method is not available in native environment
```

Parameters

`text`

A string to put into a port's input queue.

`encoding`

String text encoding. Can be either a member of Common.Encoding enumeration or a numeric Windows code page identifier. If omitted, defaults to UTF8 encoding.

Description

Put a given string into port's input queue. An optional `encoding` parameter, if specified, tells how the port should encode a passed string.

---

provideReceivedData

```
TypeScript
provideReceivedData(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 put into a port's input queue.

Description

Put a single byte into port's input queue.

---

provideReceivedData

```
TypeScript
provideReceivedData(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`

An array, typed array, array buffer or `DataView` object to put into port's input queue.

Description

Put a given data array into port's input queue.

provideReceivedData

```TypeScript
provideReceivedData(sendAs: Port.WriteAs,
    data: number[] | Uint8Array | Uint16Array | Uint32Array | ArrayBuffer | DataView,
    bigEndian?: boolean): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`sendAs`

How to interpret the passed array.

`data`

An array, typed array, array buffer or `DataView` object to put into port's input queue. `sendAs` parameter is used to interpret and possibly convert the passed data array.

`bigEndian`

If `sendAs` parameter describes multi-byte sequence, treat is as little-endian (`bigEndian` equals `false` or omitted) or as big-endian (`bigEndian` equals `true`).

Description

Put a given data array into port's input queue.

---

getSentData

```TypeScript
getSentData(maxBytes?: number): Promise<Uint8Array>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`maxBytes`

An optional maximum size of the returned buffer.

Return Value

A promise that is resolved with a copy of the data from the output queue.

Description

Request the data that have been put into the port's *output queue* by the application. This method must be used by a device script utilizing a pull model.

setError

```TypeScript
setError(errorCode: number): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`errorCode`

A Win32 error code.

Description

Set the port into the error state. Any pending read sent by application is immediately completed with a passed error code.

---

setEventMask

```TypeScript
setEventMask(mask: Port.EventMask): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`mask`

Event mask to set. Can be a combination of flags in Port.EventMask enumeration.

Description

Set the port event mask.

---

clearEventMask

```TypeScript
clearEventMask(): void;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Description

Clear the port event mask.

IScriptDevice Interface

Description

This interface must be implemented by a custom device script.

> **NOTE**
>
> This interface consists of optional members. A device script utilizing a pull model and not requiring custom parameter support will actually "inherit" none of this interface members.

Declaration

TypeScript

```typescript
interface IScriptDevice {

    // Methods
    onSend(data: Uint8Array): Promise<void>;
    setParam(name: string, value: string): void;
}
```

C#

```csharp
public interface IScriptDevice
{

    // Methods
}
```

C++

```cpp
struct IScriptDevice : IDispatch
{

    // Methods
};
```

IScriptDevice Methods

---

onSend

TypeScript

```typescript
onSend(data: Uint8Array): Promise<void>;
```

C#

```csharp
// This method is not available in managed environment
```

C++

```cpp
// This method is not available in native environment
```

Parameters

`data`

A copy of the data from the port's output queue.

Description

This optional method must be implemented by a device script utilizing a push model. Application's write request will not be completed until the promise returned by this method is resolved.

---

setParam

```
TypeScript
setParam(name: string, value: string): void;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`name`

A custom parameter name.

`value`

A custom parameter value.

Description

This optional method must be implemented by a device script that wants to provide external code a way to customize its behavior.

An external caller may either use the IScriptPortDevice.setScriptParam method or send a custom IOCTL_SCRIPTPORT_SET_PARAM I/O control request to an opened port.

Common.Encoding Enumeration

Description

Predefined text encodings.

Symbol	Value	Description
`Utf8`	`-1`	UTF8 encoding.
`Ascii`	`-2`	ASCII (7-bit) encoding.
`Utf16LE`	`-3`	UTF16 little-endian encoding.
`Utf16BE`	`-4`	UTF16 little-endian encoding.

Port.WriteAs Enumeration

Description

Data encoding override.

Symbol	Value	Description
`Bytes`	`0`	Encode data array as byte array
`Words`	`1`	Encode data array as word array
`DoubleWords`	`2`	Encode data array as double word array

Port.EventMask Enumeration

Description

A serial port event mask.

Symbol	Value	Description
`Break`	`0x0040`	A break was detected on input
`CTS`	`0x0008`	The CTS (clear-to-send) signal changed state
`DSR`	`0x0010`	The DSR (data-set-ready) signal changed state
`Error`	`0x0080`	A line - status error occurred.Line - status errors are `CE_FRAME`, `CE_OVERRUN`, and `CE_RXPARITY`
`Ring`	`0x0100`	A ring indicator was detected
`RLSD`	`0x0020`	The RLSD (receive-line-signal-detect) signal changed state
`RxChar`	`0x0001`	A character was received and placed in the input buffer
`RxFlag`	`0x0002`	The event character was received and placed in the input buffer.
`TxEmpty`	`0x0004`	The last character in the output buffer was sent

File system API

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 {
    // Properties
    readonly tempFolder: string;
    readonly windowsFolder: string;
    readonly systemFolder: string;
    readonly programDataFolder: string;

    // Methods
    createFile(path: string,
        openMode: FS.OpenMode,
        access: FS.Access,
        share?: FS.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;
    loadTextFile(path: string, utf8?: boolean): string;
}
```

```C#
// This interface is not available in managed environment
```

```C++
// This interface is not available in native environment
```

IFileManager Properties

---

tempFolder

```TypeScript
readonly tempFolder: string;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Get the system temporary folder. Usually equals to `c:\Windows\TEMP`.

---

windowsFolder

```TypeScript
readonly windowsFolder: string;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Get the OS root folder. Usually equals to `c:\Windows`.

---

systemFolder

```TypeScript
readonly systemFolder: string;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

Get the OS system folder. Usually equals to `c:\Windows\System32`.

---

programDataFolder

```
TypeScript
readonly programDataFolder: string;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Get the `ProgramData` folder. Usually equals to `c:\ProgramData`.

IFileManager Methods

---

createFile

```
TypeScript
createFile(path: string,
    openMode: FS.OpenMode,
    access: FS.Access,
    share?: FS.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 FS.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 `FS.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", FS.OpenMode.OpenExisting,
FS.Access.Read, FS.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.

---

loadTextFile

```TypeScript
loadTextFile(path: string, utf8?: boolean): 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 file.

`utf8`

Optional flag to force the file to be treated as UTF8-encoded. Defaults to `true` if omitted.

Description

This method attempts to load and parse a text file. It tries to automatically determine the text file encoding by analyzing its structure and presence of BOM at the beginning of a file.

IFile Interface

Description

This interface is implemented by a file object. File objects are obtained by calling a IFileManager.createFile method.

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.

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.

FS.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 `FS.Access.Write` access specifier.

FS.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.

FS.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.

Network API

INetworkManager Interface

Description

This interface is implemented by a Network manager object. A reference to this object is obtained using the global net property.

Declaration

```TypeScript
interface INetworkManager {

    // Methods
    createTcpSocket(bufferSize?: number): ITcpSocket;
    createUdpSocket(): IUdpSocket;
    createTcpListener(bufferSize?: number): ITcpListener;
}
```

```C#
// This interface is not available in managed environment
```

```C++
// This interface is not available in native environment
```

INetworkManager Methods

---

createTcpSocket

```TypeScript
createTcpSocket(bufferSize?: number): ITcpSocket;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`bufferSize`

Optional size of the socket buffer, in bytes. If omitted, 512 KB buffer is used.

Return Value

Reference to a created TCP socket object.

Description

Create a TCP socket.

---

createUdpSocket

```TypeScript
createUdpSocket(): IUdpSocket;
```

```C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Return Value

Reference to a created UDP socket object.

Description

Create an UDP socket.

> **WARNING**
>
> This method is not available on Windows 7 and throws an exception.

---

createTcpListener

```
TypeScript
createTcpListener(bufferSize?: number): ITcpListener;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`bufferSize`

Optional size of the socket buffer, in bytes. If omitted, 512 KB buffer is used.

Return Value

A reference to a create TCP listener object.

Description

Create a TCP listener. A listener may be used to wait for incoming TCP connections.

ITcpSocket Interface

Description

This interface is implemented by a TCP socket. Call the INetworkManager.createTcpSocket method to create a TCP socket.

This interface extends the Net.ISocket and does not add any additional properties or methods.

Declaration

```
TypeScript
interface ITcpSocket extends Net.ISocket {
}
```

```C#
// This interface is not available in managed environment
```

```C++
// This interface is not available in native environment
```

IUdpSocket Interface

Description

This interface is implemented by an UDP socket. Call theINetworkManager.createUdpSocket method to create an UDP socket.

> **WARNING**
>
> This functionality is not available on Windows 7. Attempt to create a UDP socket with a call to INetworkManager.createUdpSocket method will fail.

Declaration

```TypeScript
interface IUdpSocket extends Net.ISocket {

    // Methods
    bind(host: string, port: number): Promise<void>;
}
```

```C#
// This interface is not available in managed environment
```

```C++
// This interface is not available in native environment
```

IUdpSocket Methods

bind

```TypeScript
bind(host: string, port: number): Promise<void>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`host`

Host name.

`port`

UDP port number.

Description

Bind an UDP socket.

ISocket Interface

Description

A base interface for both ITcpSocket and IUdpSocket objects.

Declaration

```TypeScript
interface ISocket {

    // Methods
    connect(host: string, port: string): Promise<void>;
    close(): void;
    send(text: string, encoding: Common.Encoding | number): Promise<number>;
    send(byte: number): Promise<number>;
    send(bytes: number[] | Uint8Array | Uint16Array | Uint32Array | ArrayBuffer | DataView):
Promise<number>;
    receive(): Promise<Uint8Array>;
}
```

```C#
// This interface is not available in managed environment
```

```C++
// This interface is not available in native environment
```

ISocket Methods

---

connect

```TypeScript
connect(host: string, port: string): Promise<void>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`host`

A host name

`port`

TCP/UDP port

Description

Connects the socket to the remote endpoint.

---

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 active connection.

---

send

```TypeScript
send(text: string, encoding: Common.Encoding | number): Promise<number>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`text` :

`encoding`

An optional encoding or Windows code page to use. If omitted, equals to `Common.Encoding.Utf8`.

Return Value

The number of bytes sent to the network.

Description

A string to send

Send the given string encoded using the specified encoding to the socket.

---

send

```TypeScript
send(byte: number): Promise<number>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`byte`

A single byte to send

Return Value

The number of bytes sent to the network.

Description

Send a single byte to the network.

---

send

```TypeScript
send(bytes: number[] | Uint8Array | Uint16Array | Uint32Array | ArrayBuffer | DataView):
Promise<number>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`bytes`

An array, typed array, array buffer or `DataView` object.

Return Value

The number of bytes sent to the network.

---

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 byte array with data received from a socket.

Description

Receive data from a socket.

Common.Encoding

ITcpListener Interface

Description

This interface is implemented by a TCP listener object. Use the INetworkManager.createTcpListener method to create one.

Declaration

```TypeScript
interface ITcpListener {

    // Methods
    bind(port: number): Promise<void>;
    bind(host: string, port: number): Promise<void>;
    listen(): Promise<ITcpSocket>;
}
```

```C#
// This interface is not available in managed environment
```

```C++
// This interface is not available in native environment
```

ITcpListener Methods

---

bind

```TypeScript
bind(port: number): Promise<void>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`port`

Local TCP port number.

Description

Bind a TCP listener to a given local TCP port on all local interfaces.

---

bind

```TypeScript
bind(host: string, port: number): Promise<void>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`host`

Local host name/address.

`port`

Local TCP port number.

Description

Bind a TCP listener to a given local endpoint.

---

listen

```TypeScript
listen(): Promise<ITcpSocket>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Return Value

When connection occurs, a connected TCP socket is returned.

Description

Listen for incoming connection. `bind` must be called before calling this method.

ITcpSocket

HTTP API

IHttpClient Interface

Description

This interface is implemented by an HTTP client object. A reference to this object is obtained using the global http property.

Declaration

```TypeScript
interface IHttpClient {

    // Methods
    request(url: string, verb: string, options?: Partial<IHttpRequestOptions>):
Promise<IHttpResponse>;
    get(url: string, options?: Partial<IHttpRequestOptions>): Promise<IHttpResponse>;
    post(url: string, options?: Partial<IHttpRequestOptions>): Promise<IHttpResponse>;
    getString(url: string, accepts?: string, options?: Partial<IHttpRequestOptions>):
Promise<string>;
    getBlob(url: string, options?: Partial<IHttpRequestOptions>): Promise<Uint8Array>;
}
```

```C#
// This interface is not available in managed environment
```

```C++
// This interface is not available in native environment
```

IHttpClient Methods

---

request

```TypeScript
request(url: string, verb: string, options?: Partial<IHttpRequestOptions>):
Promise<IHttpResponse>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`url`

Resource url.

`verb`

HTTP verb to use, such as “GET”, “POST”, “HEAD” and so on.

`options`

An optional request options. Allows you to set request body and headers.

Return Value

An HTTP response object.

Description

Make an HTTP request. All other methods of this interface eventually call this method. Prefer using other methods unless you want to customize an HTTP request properties.

---

get

```
TypeScript
get(url: string, options?: Partial<IHttpRequestOptions>): Promise<IHttpResponse>;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`url`

Resource url.

`options`

An optional request options. Allows you to set request body and headers.

Return Value

An HTTP response object.

Description

Send an HTTP GET request and obtain a response. Calls the request method.

---

post

```
TypeScript
post(url: string, options?: Partial<IHttpRequestOptions>): Promise<IHttpResponse>;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`url`

Resource url.

`options`

An optional request options. Allows you to set request body and headers.

Return Value

An HTTP response object.

Description

Send an HTTP POST request and obtain a response. Calls the request method.

---

getString

```TypeScript
getString(url: string, accepts?: string, options?: Partial<IHttpRequestOptions>):
Promise<string>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`url`

Resource url.

`accepts`

A value of `Accepts` header. Defaults to `application/json`.

`options`

An optional request options. Allows you to set request body and headers.

Return Value

Contents of a remote resource as a string.

Description

Send an HTTP GET request and returns the result as a string. Calls the request method.

---

getBlob

```TypeScript
getBlob(url: string, options?: Partial<IHttpRequestOptions>): Promise<Uint8Array>;
```

```C#
// This method is not available in managed environment
```

```C++
// This method is not available in native environment
```

Parameters

`url`

Resource url.

`options`

An optional request options. Allows you to set request body and headers.

Return Value

Contents of a remote resource as a byte array.

Description

Send an HTTP GET request and returns the result as a string. Calls the request method.

IHttpRequestOptions Interface

Description

An object with this interface is usually created by a device script and passed to IHttpClient.request method to customize its behavior.

Declaration

```
TypeScript
interface IHttpRequestOptions {
    // Properties
    headers?: string[] | IHttpHeader[];
    body?: string | number[] | Uint8Array;
    encoding?: UnicodeEncoding;
    mediaType?: string;
}
```

```
C#
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

IHttpRequestOptions Properties

---

headers

```
TypeScript
headers?: string[] | IHttpHeader[];
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

HTTP headers, either as a string array or array of IHttpHeader objects. If string array is used, each string must be of the form `Name: Value`.

---

body

```
TypeScript
body?: string | number[] | Uint8Array;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

HTTP request body, can be a string, array or a byte array

---

encoding

```
TypeScript
encoding?: UnicodeEncoding;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Encoding for string body. Used only if `body` is a string. Defaults to UTF8.

---

mediaType

```
TypeScript
mediaType?: string;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

Body media type. Defaults to `text/plain`. Used only if `body` is a string. Defaults to UTF8.

IHttpHeader Interface

```
TypeScript
interface IHttpHeader {
    // Properties
    name: string;
    value: string;
}
```

```
C#
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

IHttpHeader Properties

---

name

```typescript
name: string;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Header name.

---

value

```typescript
value: string;
```

```csharp
// This property is not available in managed environment
```

```cpp
// This property is not available in native environment
```

Description

Header value.

IHttpResponse Interface

Description

This interface is implemented by an HTTP response object, returned by methods of IHttpClient interface.

Declaration

```typescript
interface IHttpResponse {
    // Properties
    readonly statusCode: number;
    readonly isSuccessful: boolean;
    readonly content: Promise<string>;
    readonly blob: Promise<Uint8Array>;
    readonly stream: Promise<IInputStream>;
}
```

```csharp
// This interface is not available in managed environment
```

```cpp
// This interface is not available in native environment
```

IHttpResponse Properties

statusCode

```TypeScript
readonly statusCode: number;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

HTTP status code.

---

isSuccessful

```TypeScript
readonly isSuccessful: boolean;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

True if `statusCode` is between 200 and 299 inclusive.

---

content

```TypeScript
readonly content: Promise<string>;
```

```C#
// This property is not available in managed environment
```

```C++
// This property is not available in native environment
```

Description

HTTP Response content, encoded as string.

---

blob

```TypeScript
readonly blob: Promise<Uint8Array>;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

HTTP Response content, as a byte array.

---

stream

```
TypeScript
readonly stream: Promise<IInputStream>;
```

```
C#
// This property is not available in managed environment
```

```
C++
// This property is not available in native environment
```

Description

HTTP Response content stream.

IInputStream Interface

Description

This interface is implemented by a stream object returned by the IHttpResponse.stream property.

Declaration

```
TypeScript
interface IInputStream {

    // Methods
    readChunk(maxSize: number): Promise<Uint8Array>;
}
```

```
C#
// This interface is not available in managed environment
```

```
C++
// This interface is not available in native environment
```

IInputStream Methods

---

readChunk

```
TypeScript
readChunk(maxSize: number): Promise<Uint8Array>;
```

```
C#
// This method is not available in managed environment
```

```
C++
// This method is not available in native environment
```

Parameters

`maxSize`

A maximum number of bytes to read.

Return Value

Read data as a byte array.

Description

Read a portion of data from the stream.

UnicodeEncoding Enumeration

Symbol	Value	Description
`Utf8`	`0`	Encode text as UTF8. This is the default value.
`Utf16LE`	`1`	Encode text as little-endian UTF16.
`Utf16BE`	`2`	Encode text as big-endian UTF16.

IOCTL_SCRIPTPORT_SET_PARAM Device I/O Request

An application that opens a virtual script port may communicate with a device script by sending a custom device control request. The numeric value of this IOCTL is `0x82FFA0C0`.

It must pass two strings with the following encoding:

`Custom Parameter Name`

Data	Encoding	Description
String length	32-bit little-endian integer	Length of a string, in characters
String data	Array of 16-bit little-endian `wchar_t`	String characters

`Custom Parameter Value`

Data	Encoding	Description
String length	32-bit little-endian integer	Length of a string, in characters
String data	Array of 16-bit little-endian `wchar_t`	String characters

Sending this IOCTL results in calling of the IScriptDevice.setParam method.

Open Source Components

Virtual Serial Port Tools uses a number of open source components. This page lists all projects used and provides their licenses. HHD Software expresses its enormous gratitude to the authors and contributors of the following projects:

TypeScript

TypeScript is a language for application-scale JavaScript. TypeScript adds optional types to JavaScript that support tools for large-scale JavaScript applications for any browser, for any host, on any OS. TypeScript compiles to readable, standards-based JavaScript.
https://github.com/microsoft/TypeScript/blob/main/LICENSE.txt

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

ChakraCore

ChakraCore is a JavaScript engine with a C API you can use to add support for JavaScript to any C or C compatible project. It can be compiled for x64 processors on Linux macOS and Windows. And x86 and ARM for Windows only. It is a future goal to support x86 and ARM processors on Linux and ARM on macOS.

ChakraCore is available under MIT (https://github.com/chakra-core/ChakraCore/blob/master/LICENSE.txt).

The MIT License (MIT)

Copyright (c) Microsoft Corporation
All rights reserved.
Copyright (c) 2021 ChakraCore Project Contributors. All rights reserved.

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.

Boost

Boost provides free peer-reviewed portable C++ source libraries. https://www.boost.org/users/license.html.

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

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, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Eigen3

Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms. https://www.mozilla.org/en-US/MPL/2.0/.