Table of Contents

Remote Serial Ports Documentation

Remote Serial Ports is a software package that provides remote access to local serial devices. It consists of two components: server and client.

Server component needs to be installed on a computer which serial ports you want to "share" across 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.

Client component is installed on a client computer. It consists of a virtual serial port driver, configuration utility and command-line utility. In addition, full-featured API is provided by means of installed in-process COM server.

Once created, a virtual serial port is connected to a remote shared physical serial port. After that, it behaves exactly like the remote port, hiding any differences from an application that works with a serial port on a client computer.

The actual connection is established at the time client application opens the virtual serial port and dropped as soon as the port is closed.

Server Documentation

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 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 over 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 (default), 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
```

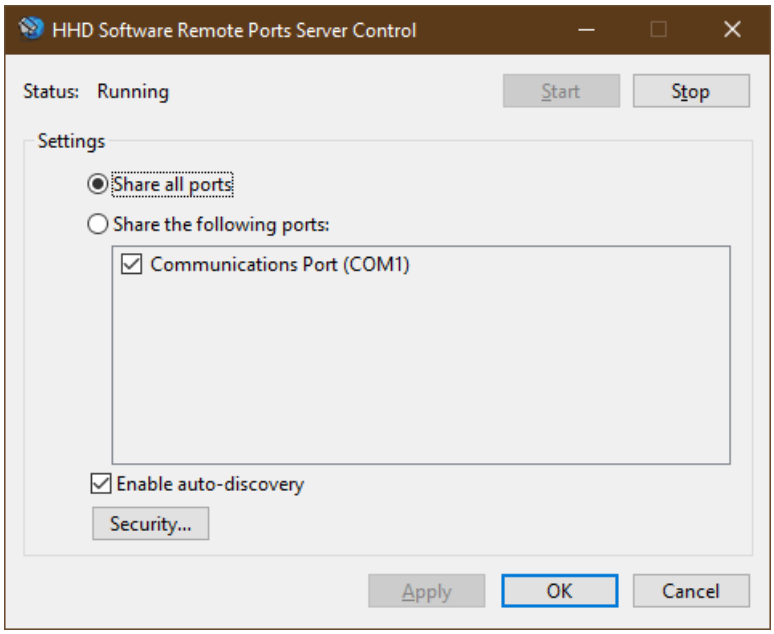
Stand-alone Mode

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 over 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.
`-uninstall-service`, `--uninstall-service`		Uninstall service.

Client Documentation

Client component consists of a virtual serial port driver, in-process COM server,configuration utility and command-line utility.

Virtual serial port driver is a user-mode driver providing full-featured serial ports to the client computer. User application may use API provided by the in-process COM server to create and delete virtual serial ports. Created virtual serial ports may then be connected to remote hosts.

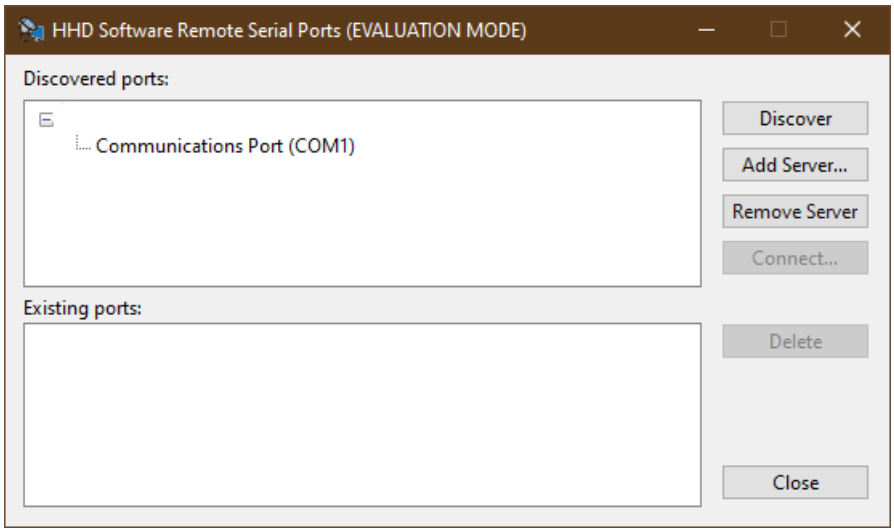
In addition, a Configuration Utility may be used to create, delete and connect virtual serial ports.

Command-line utility also provides a way to create, delete and list configured remote serial ports from using the command-line interface.

When the client application opens a handle to a virtual serial port, a connection is established to remote host. All control information and data sent or received to the virtual serial port after that is silently transferred to the server and then eventually to the shared server port.

Client Configuration Utility

Client configuration utility is a convenient way to create and manage local virtual serial ports.



The top list displays all discovered or manually added servers and their shared serial ports. Press the **Discover** button to automatically search for serial port servers in the local network. In addition, application automatically starts discovering upon launch. If the target server is not discovered automatically, press the **Add Server** button to add it manually. Press the **Remove Server** button to remove a server from the list. Select the port and press the **Connect** button to create a local virtual serial port and connect it to the selected remote port.

The bottom list displays all currently configured local ports and their remote connection endpoints.

When connecting new serial port, you are asked for the credentials. These credentials are used to authenticate on the remote server each time a client application opens a handle to a virtual serial port and a connection is established.

Press the **Delete** button to remove the local serial port.

Command-Line Utility

Command-line utility **rspcli.exe** may be used as a simple API to create and manage remote serial ports. Command-line utility returns 0 if the requested operation 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	Description
`-?, --help`		Displays the list of supported parameters with short description.
`--silent`		Do not display any error or success messages.
`-create`		Create new port and connect it to remote port.
`-delete`	`N`	Delete an existing port.
`-list`		List all remote serial ports on the current computer.
`-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.
`--remote-host`	`hostname`	Name or address of remote host that shares COM port.
`--remote-port`	`N`	COM port number of the port on remote computer.
`--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.

Examples

Create New Remote Port

The following command-line creates new remote serial port. The user will be asked to enter password in the terminal:

```
Command Prompt
rspcli.exe -create --remote-host server --remote-port 1 --login domain\user
```

The following command-line creates new remote serial port and assigns a specific COM port number to it:

```
Command Prompt
rspcli.exe -create --local-port 2 --remote-host server --remote-port 1 --login domain\user --
```

Delete Remote Serial Port

The following command-line deletes a previously created remote serial port:

```
Command Prompt
rspcli.exe -delete 2
```

List All Remote Serial Ports

The following command-line lists all currently configured serial ports with their parameters:

```
Command Prompt
rspcli.exe -list
```

List All Shared Serial Ports

The following command-line lists all ports that are shared by the remote host:

```
Command Prompt
rspcli.exe -list-remote server
```

API

Remote Serial Ports client 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, first use the `#import` directive to bring definitions into the scope:

```
C++
#import "full_path_to_hhdclientctl.dll"
```

or

```
C++
#import "progid:hhdclientctl.RemotePortLibrary.1"
```

or

```
C++
#import "libid:C18B7B8C-07A8-4CA8-B19F-91D79F8F099F" version("1.0")
```

This will scan the type library and generate wrappers for interfaces and enumerations.

Then, use the generated definitions. For example, to create an instance of the library object, use the following code:

```
C++
hhdclientctl::IRemotePortLibraryPtr pLibrary;
if (SUCCEEDED(pLibrary.CreateInstance(__uuidof(hhdclientctl::RemotePortLibrary))))
{
	// create new virtual serial port with automatically-assigned port name
	auto pNewPort1 = pLibrary->createPort();
	// manually tell which port to create
	auto pNewPort2 = pLibrary->createPort(L"COM10");
	// configure created port
	pNewPort1->put_remoteHost = L"hostname";
	// ...
}
```

Using from C#

In order to use the library from C# project, Right-click the References project folder in Microsoft Visual Studio and add a reference to the `hhdclientctl.dll` library. After this is done, add the `using` declaration to the source file:

```
C#
using hhdclientctl;
```

Use the API provided by the library:

```
C#
var library = new hhdclientctl.IRemotePortLibrary();
var port1 = library.createPort();
port1.remoteHost = "remote.hostname";
port1.remotePort = 5;
port1.login = "username";
port1.password = "password";
port1.domain = "domain";
// ...
```

Using from JavaScript

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

```
JavaScript
var library = new ActiveXObject("hhdclientctl.RemotePortLibrary.1");
```

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

```
JavaScript
// create new virtual serial port
var port1 = library.createPort();
// manually tell which port to create
var port2 = library.createPort("COM10");
// configure serial port
port1.remoteHost = "remotehostname";
port1.remotePort = 5;
// ...
```

Using from TypeScript

Remote Serial Ports library contains TypeScript definition files that simplifies library usage in scripting environments. It automatically gives you method parameter type validation and simplifies specifying callback methods.

```typescript
///<reference path="clientctl.d.ts" />
var library = (IRemotePortLibrary) new ActiveXObject("hhdclientctl.RemotePortLibrary.1");
// create new virtual serial port with automatically-assigned port name
var port1 = library.createPort();
// manually tell which port to create
var port2 = library.createPort("COM10");
// configure port
port1.remoteHost = "remotehostname";
port1.remotePort = 5;
// ...
```

IRemotePortLibrary Interface

```typescript
interface IRemotePortLibrary {
    // Methods
    createPort(port?: number | string): IRemotePortDevice;
    getPortsJs(): IRemotePortDevice[];
    getRemoteSharedPortsJs(hostName: string): IRemotePortDescription[];
    installLicenseFile(path: string): void;
}
```

```csharp
public interface IRemotePortLibrary
{
    // Methods
    IRemotePortDevice createPort(object port);
    Array getPorts();
    Array getRemoteSharedPorts(string hostName);
    void installLicenseFile(string path);
    void installLicenseInMemory(byte[] data);
}
```

```cpp
struct IRemotePortLibrary : IDispatch
{
    // Methods
    IRemotePortDevicePtr createPort(const _variant_t & port);
    SAFEARRAY(IRemotePortDevice) getPorts();
    SAFEARRAY(IRemotePortDescription) getRemoteSharedPorts(_bstr_t hostName);
    HRESULT installLicenseFile(_bstr_t path);
    HRESULT installLicenseInMemory(SAFEARRAY(BYTE) data);
};
```

IRemotePortLibrary Methods

createPort

```typescript
createPort(port?: number | string): IRemotePortDevice;
```

```csharp
IRemotePortDevice createPort(object port);
```

```cpp
IRemotePortDevicePtr createPort(const _variant_t & port);
```

Parameters

`port`

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

Return Value

The method returns a reference to created port device.

Description

Create new virtual serial port.

getPorts

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

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

C++
```
SAFEARRAY(IRemotePortDevice) getPorts();
```

Return Value

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

getPortsJs

TypeScript
```
getPortsJs(): IRemotePortDevice[];
```

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

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

Return Value

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

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.

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.

IRemotePortDevice Interface

Description

`IRemotePortDevice` interface is implemented by the virtual serial device object. You can use properties and methods of this interface to control the device and connect it to remote port.

Declaration

```
TypeScript
interface IRemotePortDevice {
    // Properties
    readonly devicePath: string;
    readonly port: number;
    remoteHost: string;
    remotePort: number;
    connectionTimeout: number;
    connectionAttempts: number;
    login: string;
    password: string;
    domain: string;
    // Methods
    deleteDevice(): void;
}
```

```
C#
public interface IRemotePortDevice
{
    // Properties
    string devicePath { get; }
    uint port { get; }
    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; }
    // Methods
    void deleteDevice();
}
```

```
C++
struct IRemotePortDevice : IDispatch
{
    // Properties
    _bstr_t devicePath;  // get
    unsigned int port;  // get
    _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
    // Methods
    HRESULT deleteDevice();
};
```

IRemotePortDevice Properties

devicePath

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

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

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

Description

Device path string. This string may be used to open the handle to the port, as an alternative to traditional `COMn` string. This property is read-only.

port

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

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

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

Description

Port number associated with this device. This property is read-only.

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.

IRemotePortDevice Methods

deleteDevice

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

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

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

Description

Deletes the associated virtual serial port. After calling this method you can no access any properties on this object. The only allowed operation is releasing interface reference.

Example

The following code snippet illustrates the creation and configuration of a 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("hhdclientctl.RemotePortLibrary.1");
    var port = library.createPort();

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

    return port.devicePath;
}
```

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.

Redistribution

Server Redistribution

Server component may be redistributed using one of the following ways:

Installer Redistribution

Server installation package may be freely re-distributed to other computers. Installer supports unattented installation:

Installation

```
Command Prompt
remote-serial-ports-server.exe -silent -machine
```

Uninstallation

```
Command Prompt
remote-serial-ports-server.exe -silent -machine -uninstall
```

In-place Upgrade

```
Command Prompt
remote-serial-ports-server.exe -silent -machine -type 2
```

Since installer adds a system service to the target machine, it must be launched by the user with sufficient privileges (and must be elevated). Any non-zero error code returned by the installer should be treated as error.

Running as system service, server always shares configured serial ports and does not require a logged-on user.

Copy-Paste Redistribution

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.

In this mode, server does not need to be elevated and may be launched by a non-privileged user. However, it requires a user to log in to the server and manually launch a server process. Ports are shared as long as server process is running.

Client Redistribution

Remote Serial Ports has a special build best suited for purposes of redistribution of the library as part of another product. You may download this package by following this link.

This package is essentially the same installer package with Configuration Utility and documentation removed. It contains both x86 and x64 versions of the library and driver and automatically installs the correct files on the target machine. On 64-bit machine, both 32-bit and 64-bit COM libraries are installed and registered.

Installer package supports unattended silent installation using the following command line switches:

Installation

```
Command Prompt
remote-serial-ports-redist.exe -silent -machine
```

Uninstallation

```
Command Prompt
remote-serial-ports-redist.exe -silent -machine -uninstall
```

In-place Upgrade

```
Command Prompt
remote-serial-ports-redist.exe -silent -machine -type 2
```

It is recommended to restart a computer after installing or uninstalling the package.

Installation Instructions

Since installer adds a device driver to the target machine, it must be launched by the user with sufficient privileges (and must be elevated). Any non-zero error code returned by the installer should be treated as error.