Table of Contents

Introduction

Device Monitoring Studio Server is a software component that provides remote access to serial, USB and network devices, connected to the computer.

A server consists of the following components:

Monitoring Modules

Monitoring modules are filter device drivers and source libraries that can be attached to any supported device and provide the caller with a copy of transferred data and commands. Monitoring modules include serial, USB and network.

Main Server Component

This component is installed as Windows Service, runs whenever the computer is running (a logged on user is not required) and provides remote access to connected devices. This component is subject for administration.

Administration MMC Snap-In

This is a standard MMC snap-in used to administer the server component.

API

Application programming interface, provided by main server component, allows administrators to use such technologies as Windows Scripting Host and PowerShell to administer and manage the server.

Installation

Device Monitoring Studio Server is distributed as stand-alone installer and may be installed on any supported version of Windows operating system.

A server may be installed on the same computer where the Device Monitoring Studio client is installed (in this case the versions of DMS Client and DMS Server must match). Although, it is not recommended to use remote connection to monitor local devices, as performance will be worse.

After installation, a server is immediately running (unless a restart is required to complete driver installation). Server does not require any user to log in, as it is installed as Windows Service. When running, a server is ready to accept connections, subject to current security settings.

Default security configuration accepts a connection from members of Everyone security group. That means that any user is allowed to connect and monitor any supported device.

Note that this will also include anonymous users, but ONLY when the server computer is configured to accept anonymous logon.

It is recommended for a server administrator to launch the administration utility immediately after installation to change the default security configuration.

Activation

Device Monitoring Studio Server operates in trial mode until it is activated. To activate, you need to obtain a license file. This section describes the ways you may use to apply the license file:

- A simplest way is to double-click the license file (license file is a file with `.dmssrvlic` extension). After the file is applied, Device Monitoring Studio Server must be restarted in order to load the new license.
- Launch the following command line:

```
PowerShell
dmssrv.exe -license <full-path-to-license-file>
```

After the file is applied, Device Monitoring Studio Server must be restarted in order to load the new license.

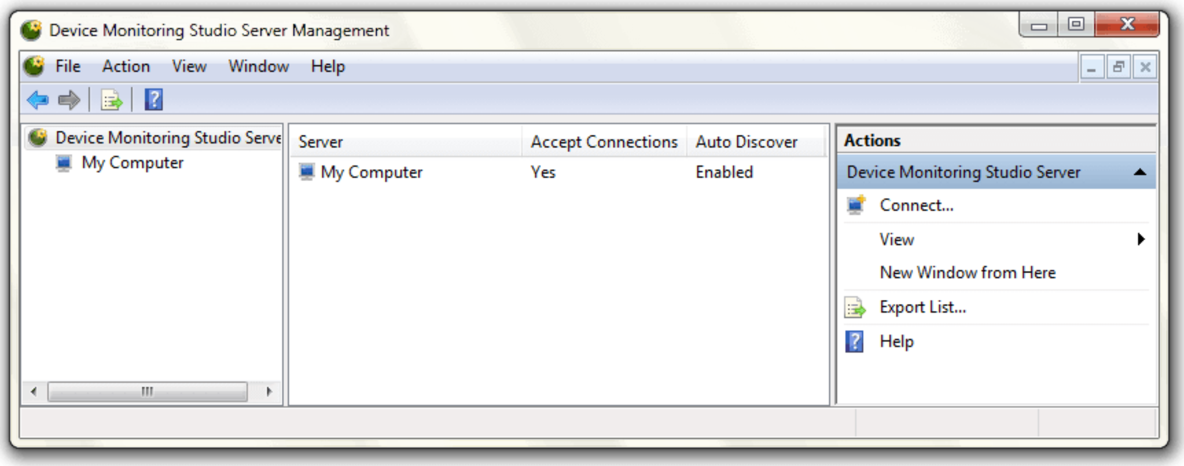
- Use the MMC Snap-In to install a license. Right-click the computer item, select "Properties". On the General page, click the Install License button and select the license file.
- Use the IMonitoringAdmin.InstallLicense method to install a license file.

Last two methods do not require you to restart the server.

Administration

Management MMC Snap-In

During installation, Device Monitoring Studio Server installs a configuration MMC Snap-In. A link to snap-in is added to Start Menu and the snap-in is also available in the list of all snap-ins.


Device Monitoring Studio Server Management
File
Action
View
Window
Help
Device Monitoring Studio Serve
My Computer
Actions
Device Monitoring Studio Server
Connect...
View
New Window from Here
Export List...
Help

Server	Accept Connections	Auto Discover
My Computer	Yes	Enabled

This snap-in is used for two purposes: administration and management.

Snap-in provides you with three types of objects:

Computers

Currently, there can be only one computer object (administration of remote servers is not supported yet).

Connections

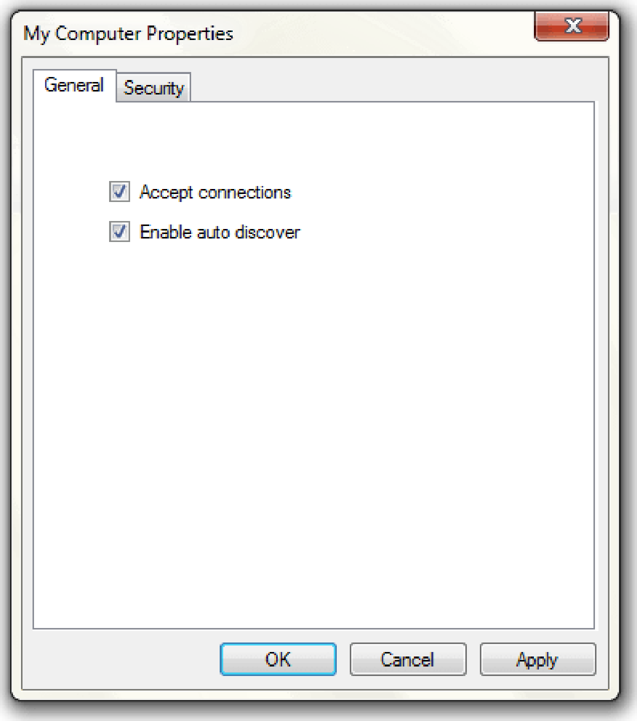
A connection represents a single client connection to the server.

Sessions

A session represents a single monitoring session from a client.

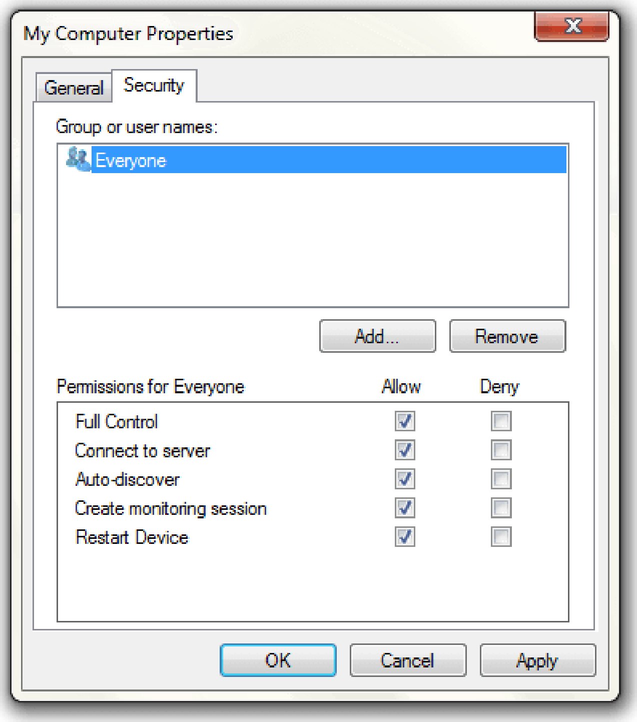
Administration

For the purposes of administration, Device Monitoring Studio Server Management MMC Snap-In provides you with the Properties command on the Computer node. After invoked, it opens the following window:



[✓] Accept connections

[✓] Enable auto discover



Group or user names:

Everyone

Permissions for Everyone	Allow	Deny
Full Control	✓	
Connect to server	✓	
Auto-discover	✓	
Create monitoring session	✓	
Restart Device	✓	

General Page

General page contains two settings: here you may control whether the server accepts connections and whether it is available for auto-discovery.

Security Page

Device Monitoring Studio Server supports Windows Security. This page allows you to configure users and groups and grant them required rights. The following permissions are defined:

Permission	Description
Connect to server	A user is allowed to establish a connection to the server. This permission must be granted for any other permission to be effective.
Auto-discover	If a user is granted this permission to the server, he will see it in the auto-discovery list in Connect to New Server window in Device Monitoring Studio client. Connect to server permission must also be granted.
Create monitoring session	A user must have this permission to start monitoring session. Connect to server permission must also be granted to him. If the user has this permission, it may monitor any supported device.
Restart Device	This permission is required to restart any supported device connected to the server. Connect to server permission is also required. Note: this permission is currently ignored and no remote restart is implemented.

Note that if you change the security settings of the server, they will become effective only when all current connections are closed. You may force this by manually dropping connections.

Management

Connection

For each connection, the following information is displayed:

User

Name of the user who established a connection.

Source

Name or address of the computer user's client is running on.

Status

Current connection status. Disconnected connections are automatically removed from the list.

Session ID

ID of the client's session on his computer.

Process ID

ID of the client's process on his computer.

Type

Shows if the client is 32-bit or 64-bit.

The following commands are provided to the administrator:

Drop Connection

Disconnect the connection and all its sessions.

Send Message...

Send a text message to the client over the connection.

Drop All Sessions

Drops all connection sessions but does not disconnect a connection.

Session

For each session, the following information is displayed:

Devices

A list of devices monitored by this session.

Status

Session status, including server and client pause.

Bytes Transferred

A number of bytes collected and transferred by this session.

Creation Time

Session creation time.

Data Source

The type of the device(s) this session monitors.

Duration

Active duration of the session.

Information is automatically refreshed every 5 seconds.

The following commands are provided to the administrator:

Drop

Drops this session.

Pause

Pauses the session. Note that a session may be paused on the server and on the client separately. A client is not notified if the session is paused on the server. The status column shows whether the session is paused on server, client, or both.

Resume

Resumes a session paused on the server.

Scripting

In addition to MMC Snap-In, the Device Monitoring Studio Server may be managed using the API it provides. The first thing a user must do is to create a management object:

```
JavaScript
var dms = new ActiveXObject("dmssrv.MonitoringSite");
```

```
PowerShell
$dms = New-Object -ComObject 'dmssrv.MonitoringSite'
```

The caller must be a member of local or domain Administrators group and must be elevated in order to connect to management object.

Events

Management object provides two events: _IMonitoringAdminEvents.OnConnectionAdded and _IMonitoringAdminEvents.OnConnectionRemoved. Connection object provides two events as well: _IConnectionEvents.OnSessionAdded and _IConnectionEvents.OnSessionRemoved.

Reference

IMonitoringAdmin Interface

Description

This is a main interface exposed by `dmssrv.MonitoringSite` object. Use the following `PROGID` to connect to the instance: `dmssrv.MonitoringSite`. Use this interface to manage the server.

Declaration

```
TypeScript
interface IMonitoringAdmin extends IDispatch {
    // Properties
    AutoDiscover: boolean;
    readonly Connections: IConnectionCollection;
    ListenConnections: boolean;
    SecurityDescriptor: string;
    // Methods
    InstallLicense(path: string): void;
}
```

```
C#
public interface IMonitoringAdmin : IDispatch
{
    // Properties
    bool AutoDiscover { get; set; }
    IConnectionCollection Connections { get; }
    bool ListenConnections { get; set; }
    string SecurityDescriptor { get; set; }
    // Methods
    void InstallLicense(string path);
}
```

```
C++
struct IMonitoringAdmin : IDispatch
{
    // Properties
    VARIANT_BOOL AutoDiscover;  // get set
    IConnectionCollectionPtr Connections;  // get
    VARIANT_BOOL ListenConnections;  // get set
    _bstr_t SecurityDescriptor;  // get set
    // Methods
    HRESULT InstallLicense(_bstr_t path);
};
```

IMonitoringAdmin Properties

AutoDiscover

```
TypeScript
AutoDiscover: boolean;
```

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

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

Description

AutoDiscover property controls whether the server answers the auto-discover broadcast requests.

Connections

```
TypeScript
readonly Connections: IConnectionCollection;
```

```
C#
IConnectionCollection Connections { get; }
```

```
C++
IConnectionCollectionPtr Connections;  // get
```

Description

This property returns the collection of connections.

Example

Obtaining list of connections:

```
PowerShell
$dms = New-Object -ComObject 'dmssrv.MonitoringSite'
$connections = dms.Connections
```

ListenConnections

```
TypeScript
ListenConnections: boolean;
```

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

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

Description

The property controls whether the server accepts connections or not.

SecurityDescriptor

```
TypeScript
SecurityDescriptor: string;
```

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

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

Description

Get or set the server security descriptor. Security descriptor must be in string format. The following table shows the values of supported permissions:

Permission	Value
`DMS_SERVER_CONNECT`	`0x00000001`
`DMS_SERVER_AUTODISCOVER`	`0x00000002`
`DMS_SERVER_CREATESESSION`	`0x00000004`
`DMS_SERVER_RESTARTDEVICE`	`0x00000008`

Note that `DMS_SERVER_CONNECT` permission must be granted for any other permission to be effective.

Only the DACL part of passed security descriptor is used. Owner/group and audit information is ignored.

Example

Changing security descriptor.

```
PowerShell
$dms = New-Object -ComObject 'dmssrv.MonitoringSite'
$dms.SecurityDescriptor = 'D:(A;;0x3;;;WD)'
```

IMonitoringAdmin Methods

InstallLicense

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

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

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

Parameters

`path`

A full path to license file.

Description

Installs a given license file.

IConnectionCollection Interface

Description

This interface is implemented by the connection collection object.

Declaration

```
TypeScript
interface IConnectionCollection extends IDispatch {
    // Properties
    Count: number;
    [Item: number]: IConnection;
}
```

```
C#
public interface IConnectionCollection : IDispatch
{
    // Properties
    int Count { get; set; }
    IConnection Item[int Index] { get; set; }
}
```

```
C++
struct IConnectionCollection : IDispatch
{
    // Properties
    long Count;  // get set
    IConnectionPtr Item(_variant_t Index);  // get set
};
```

IConnectionCollection Properties

Count

```
TypeScript
Count: number;
```

```
C#
int Count { get; set; }
```

```
C++
long Count;  // get set
```

Description

Returns the number of connections in the collection.

Item

```
TypeScript
[Item: number]: IConnection;
```

```
C#
IConnection Item[int Index] { get; set; }
```

```
C++
IConnectionPtr Item(_variant_t Index);  // get set
```

Description

Returns the connection from the collection. Index must be an integer number. In most languages, this method may be called using the array indexing operator.

Example

Obtaining the last connection:

```
PowerShell
$dms = New-Object -ComObject 'dmssrv.MonitoringSite'
$connections = $dms.Connections
$connection = $connections[0]
```

IConnection Interface

Description

This interface is implemented by the connection object. Use it to control the individual client connection.

Declaration

```
TypeScript
interface IConnection extends IDispatch {
    // Properties
    readonly Client64: boolean;
    readonly Id: number;
    readonly ProcessId: number;
    readonly SessionId: number;
    readonly Sessions: ISessionCollection;
    readonly Source: string;
    readonly Status: ConnectionStatus;
    readonly UserName: string;
    // Methods
    Drop(): void;
    DropAllSessions(): void;
    SendCustomMessage(message: string): void;
}
```

```
C#
public interface IConnection : IDispatch
{
    // Properties
    bool Client64 { get; }
    ulong Id { get; }
    uint ProcessId { get; }
    uint SessionId { get; }
    ISessionCollection Sessions { get; }
    string Source { get; }
    ConnectionStatus Status { get; }
    string UserName { get; }
    // Methods
    void Drop();
    void DropAllSessions();
    void SendCustomMessage(string message);
}
```

```
C++
struct IConnection : IDispatch
{
    // Properties
    VARIANT_BOOL Client64;  // get
    unsigned __int64 Id;  // get
    unsigned long ProcessId;  // get
    unsigned long SessionId;  // get
    ISessionCollectionPtr Sessions;  // get
    _bstr_t Source;  // get
    ConnectionStatus Status;  // get
    _bstr_t UserName;  // get
    // Methods
    HRESULT Drop();
    HRESULT DropAllSessions();
    HRESULT SendCustomMessage(_bstr_t message);
};
```

IConnection Properties

Client64

```
TypeScript
readonly Client64: boolean;
```

```
C#
bool Client64 { get; }
```

```
C++
VARIANT_BOOL Client64;  // get
```

Description

This property holds `true` if the client is running a 64-bit operating system and false otherwise.

Id

```
TypeScript
readonly Id: number;
```

```
C#
ulong Id { get; }
```

```
C++
unsigned __int64 Id;  // get
```

Description

This property holds a unique identifier of a connection. This identifier is guaranteed to be unique for this server only.

ProcessId

```
TypeScript
readonly ProcessId: number;
```

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

```
C++
unsigned long ProcessId;  // get
```

Description

This property holds a process id of the user who established a connection.

SessionId

```
TypeScript
readonly SessionId: number;
```

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

```
C++
unsigned long SessionId;  // get
```

Description

This property holds a session id of the user who established a connection.

Sessions

```
TypeScript
readonly Sessions: ISessionCollection;
```

```
C#
ISessionCollection Sessions { get; }
```

```
C++
ISessionCollectionPtr Sessions;  // get
```

Description

Returns a collection of all connected sessions.

Source

```
TypeScript
readonly Source: string;
```

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

```
C++
_bstr_t Source;  // get
```

Description

This property returns the computer name a connection originates from.

Status

```
TypeScript
readonly Status: ConnectionStatus;
```

```
C#
ConnectionStatus Status { get; }
```

```
C++
ConnectionStatus Status;  // get
```

Description

Returns a connection status. One of the following values is returned:

Status	Value	Description
`ClientDisconnected`	`0`	Client has already been disconnected and connection is closed. Connections are never re-used.
`ClientConnected`	`1`	Client is connected.

UserName

```
TypeScript
readonly UserName: string;
```

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

```
C++
_bstr_t UserName;  // get
```

Description

This property holds a user name of the user that established a connection.

IConnection Methods

Drop

```
TypeScript
Drop(): void;
```

```
C#
void Drop();
```

```
C++
HRESULT Drop();
```

Description

Drops the current connection and all its monitoring sessions.

DropAllSessions

```
TypeScript
DropAllSessions(): void;
```

```
C#
void DropAllSessions();
```

```
C++
HRESULT DropAllSessions();
```

Description

Forcibly closes all active monitoring sessions for this connection.

SendCustomMessage

```
TypeScript
SendCustomMessage(message: string): void;
```

```
C#
void SendCustomMessage(string message);
```

```
C++
HRESULT SendCustomMessage(_bstr_t message);
```

Parameters

`message`

A text message to send.

Description

This method sends a supplied text message to the client. A client will see this message in a pop-up window. An administrator may use this method to warn clients of the forthcoming server shutdown or session drop, for example.

Example

Warning the user of the forthcoming shutdown.

```
PowerShell
$conn.SendCustomMessage('Prepare for disconnect. We are about to restart a server.')
```

ISessionCollection Interface

Description

This interface is implemented by the session collection object.

Declaration

```
TypeScript
interface ISessionCollection extends IDispatch {
    // Properties
    Count: number;
    [Item: number]: ISession;
}
```

```
C#
public interface ISessionCollection : IDispatch
{
    // Properties
    int Count { get; set; }
    ISession Item[int Index] { get; set; }
}
```

```
C++
struct ISessionCollection : IDispatch
{
    // Properties
    long Count;  // get set
    ISessionPtr Item(_variant_t Index);  // get set
};
```

ISessionCollection Properties

Count

```
TypeScript
Count: number;
```

```
C#
int Count { get; set; }
```

```
C++
long Count;  // get set
```

Description

Returns the number of sessions in the collection.

Item

```
TypeScript
[Item: number]: ISession;
```

```
C#
ISession Item[int Index] { get; set; }
```

```
C++
ISessionPtr Item(_variant_t Index);  // get set
```

Description

Returns the session from the collection. Index must be an integer number. In most languages, this method may be called using the array indexing operator.

Example

Obtaining the first connected session:

```
PowerShell
$dms = New-Object -ComObject 'dmssrv.MonitoringSite'
$connections = $dms.Connections
$connection = $connections[0]
$session = $connection.Sessions[0]
```

ISession Interface

Description

This interface is used to manage a monitoring session.

Declaration

```
TypeScript
interface ISession extends IDispatch {
    // Properties
    readonly BytesTransferred: number;
    readonly Connection: IConnection;
    readonly CreationTime: Date;
    readonly DataSource: string;
    readonly Devices: IDeviceCollection;
    readonly Id: number;
    readonly Status: SessionStatus;
    // Methods
    Drop(): void;
    Pause(): void;
    Resume(): void;
}
```

```
C#
public interface ISession : IDispatch
{
    // Properties
    ulong BytesTransferred { get; }
    IConnection Connection { get; }
    DateTime CreationTime { get; }
    string DataSource { get; }
    IDeviceCollection Devices { get; }
    ulong Id { get; }
    SessionStatus Status { get; }
    // Methods
    void Drop();
    void Pause();
    void Resume();
}
```

```
C++
struct ISession : IDispatch
{
    // Properties
    unsigned __int64 BytesTransferred;  // get
    IConnectionPtr Connection;  // get
    Date CreationTime;  // get
    _bstr_t DataSource;  // get
    IDeviceCollectionPtr Devices;  // get
    unsigned __int64 Id;  // get
    SessionStatus Status;  // get
    // Methods
    HRESULT Drop();
    HRESULT Pause();
    HRESULT Resume();
};
```

ISession Properties

BytesTransferred

```
TypeScript
readonly BytesTransferred: number;
```

```
C#
ulong BytesTransferred { get; }
```

```
C++
unsigned __int64 BytesTransferred;  // get
```

Description

This property holds the total number of transferred bytes for the current session.

Connection

```
TypeScript
readonly Connection: IConnection;
```

```
C#
IConnection Connection { get; }
```

```
C++
IConnectionPtr Connection;  // get
```

Description

This property returns a reference to a session's connection object.

CreationTime

```
TypeScript
readonly CreationTime: Date;
```

```
C#
DateTime CreationTime { get; }
```

```
C++
Date CreationTime;  // get
```

Description

This property holds the session creation time.

DataSource

```
TypeScript
readonly DataSource: string;
```

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

```
C++
_bstr_t DataSource;  // get
```

Description

Returns the type of the source for the current session. Can be `Serial`, `USB` or `Network`.

Devices

```
TypeScript
readonly Devices: IDeviceCollection;
```

```
C#
IDeviceCollection Devices { get; }
```

```
C++
IDeviceCollectionPtr Devices;  // get
```

Description

Returns a collection of session devices.

Id

```
TypeScript
readonly Id: number;
```

```
C#
ulong Id { get; }
```

```
C++
unsigned __int64 Id;  // get
```

Description

This property holds a unique identifier of a session. This identifier is guaranteed to be unique for this server only.

Status

```
TypeScript
readonly Status: SessionStatus;
```

```
C#
SessionStatus Status { get; }
```

```
C++
SessionStatus Status;  // get
```

Description

This property holds the current session status. It equals one or more of the following values:

Status	Value	Description
`SessionDisconnected`	`0x00000000`	A session has been disconnected. Session objects are never re-used.
`SessionPausedByClient`	`0x00000001`	A session has been paused by the client.
`SessionPausedByServer`	`0x00000002`	A session has been paused by the server.
`SessionRunning`	`0x00000004`	A session is running.

ISession Methods

Drop

```
TypeScript
Drop(): void;
```

```
C#
void Drop();
```

```
C++
HRESULT Drop();
```

Description

Disconnect the current session.

Pause

```
TypeScript
Pause(): void;
```

```
C#
void Pause();
```

```
C++
HRESULT Pause();
```

Description

Pauses the session. ISession.Status property will have `SessionPausedByServer` flag set after this method is called. Use the ISession.Resume method to resume the paused session.

Resume

```
TypeScript
Resume(): void;
```

```
C#
void Resume();
```

```
C++
HRESULT Resume();
```

Description

Resumes a paused session. It removes the `SessionPausedByServer` status, but cannot remove the `SessionPausedByClient` status.

IConnection

IDeviceCollection Interface

Description

This interface is implemented by the device collection object.

Declaration

```
TypeScript
interface IDeviceCollection extends IDispatch {
    // Properties
    Count: number;
    [Item: number]: IDevice;
}
```

```
C#
public interface IDeviceCollection : IDispatch
{
    // Properties
    int Count { get; set; }
    IDevice Item[int Index] { get; set; }
}
```

```
C++
struct IDeviceCollection : IDispatch
{
    // Properties
    long Count;  // get set
    IDevicePtr Item(_variant_t Index);  // get set
};
```

IDeviceCollection Properties

Count

```
TypeScript
Count: number;
```

```
C#
int Count { get; set; }
```

```
C++
long Count;  // get set
```

Description

Returns the number of devices in the collection.

Item

```
TypeScript
[Item: number]: IDevice;
```

```
C#
IDevice Item[int Index] { get; set; }
```

```
C++
IDevicePtr Item(_variant_t Index);  // get set
```

Description

Returns the device from the collection. Index must be an integer number. In most languages, this method may be called using the array indexing operator.

IDevice Interface

Description

Use this interface to query for session's device names.

Declaration

```
TypeScript
interface IDevice extends IDispatch {
    // Properties
    readonly Key: string;
    readonly Name: string;
}
```

```
C#
public interface IDevice : IDispatch
{
    // Properties
    string Key { get; }
    string Name { get; }
}
```

```
C++
struct IDevice : IDispatch
{
    // Properties
    _bstr_t Key;  // get
    _bstr_t Name;  // get
};
```

IDevice Properties

Key

```
TypeScript
readonly Key: string;
```

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

```
C++
_bstr_t Key;  // get
```

Description

This property holds a device key. A device key is a system identifier that uniquely describes the instance of the connected device. The format is internal and may be different for different device types. Note that the same device connected to another port usually generates another device key, therefore, is considered as another device by Device Monitoring Studio Server.

Name

```
TypeScript
readonly Name: string;
```

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

```
C++
_bstr_t Name;  // get
```

Description

This property holds a device name.

_IMonitoringAdminEvents Interface

Description

This interface encapsulates two events generated by monitoring site object. You implement this interface either directly or indirectly by binding to monitoring site object events.

Declaration

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

```
C#
public interface _IMonitoringAdminEvents : IDispatch
{
    // Methods
    void OnConnectionAdded(IConnection connection);
    void OnConnectionRemoved(IConnection connection);
}
```

```
C++
struct _IMonitoringAdminEvents : IDispatch
{
    // Methods
    HRESULT OnConnectionAdded(IConnection connection);
    HRESULT OnConnectionRemoved(IConnection connection);
};
```

_IMonitoringAdminEvents Methods

OnConnectionAdded

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

```
C#
void OnConnectionAdded(IConnection connection);
```

```
C++
HRESULT OnConnectionAdded(IConnection connection);
```

Parameters

`connection`

Reference to new connection object.

Description

This event is generated when new connection is established.

OnConnectionRemoved

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

```
C#
void OnConnectionRemoved(IConnection connection);
```

```
C++
HRESULT OnConnectionRemoved(IConnection connection);
```

Parameters

`connection`

Reference to disconnected connection object.

Description

This event is generated when the connection is disconnected.

IConnection

_IConnectionEvents Interface

Description

This interface encapsulates two session-related events exposed by connection object. You implement this interface either directly or indirectly by binding to connection object events.

Declaration

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

```
C#
public interface _IConnectionEvents : IDispatch
{
    // Methods
    void OnSessionAdded(ISession session);
    void OnSessionRemoved(ISession session);
}
```

```
C++
struct _IConnectionEvents : IDispatch
{
    // Methods
    HRESULT OnSessionAdded(ISession session);
    HRESULT OnSessionRemoved(ISession session);
};
```

_IConnectionEvents Methods

OnSessionAdded

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

```
C#
void OnSessionAdded(ISession session);
```

```
C++
HRESULT OnSessionAdded(ISession session);
```

Parameters

`session`

Reference to new session object.

Description

This event is fired each time new session is created.

OnSessionRemoved

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

```
C#
void OnSessionRemoved(ISession session);
```

```
C++
HRESULT OnSessionRemoved(ISession session);
```

Parameters

`session`

Reference to a disconnected session object.

Description

This event is fired when a session is disconnected.

ISession