



ServerManager

Version 3.3.0.2

2009/8/31

ServerManager

© 2009 VIVOTEK Inc. All Right Reserved

VIVOTEK may make changes to specifications and product descriptions at any time, without notice.

The following is trademarks of VIVOTEK Inc., and may be used to identify VIVOTEK products only: VIVOTEK. Other product and company names contained herein may be trademarks of their respective owners.

No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from VIVOTEK Inc.

# TABLE OF CONTENTS

## 1.OVERVIEW

1.1. Introduction .....	5
1.2. File Structure.....	5

## 2.PROGRAMMER'S GUIDE

2.1. Using ServerChannel Module .....	7
Create an Access to the Camera Server .....	7

## 3.Sample Code

3.1. Get/Set Network Information.....	9
3.2. Send PTZ Command .....	12
3.3. Get/Set DO.....	13
3.4. Send HTTP Command to Server (Non-Blocking Mode) .....	15

## 4.API Reference

4.1. Enumeration.....	18
EJOYSTICK_ZOOMING_COMMAND.....	19
ESrvCtrlDiDoLevel.....	20
ESrvCtrlPTZType.....	21
ESRVCTRL_PTZ_COMMAND.....	22
4.2. Callback Function.....	25
SVRCTRL_STATUS_CALLBACK.....	26
4.3. Data Structure .....	27
TSRVCTRL_DEV_PROPERTY.....	28
TSRVCTRL_DEV_PROPERTYW .....	29
TSRVCTRL_MODETECT_INFO .....	30
TSRVCTRL_PRIVATEMASK_INFO .....	32
TSVRCTRL_NOTIFY.....	33
TSVRCTRL_NOTIFY_CONT.....	34
TSRVMNGR_DEV_PROPERTY .....	35
TSRVMNGR_DEV_PROPERTYW.....	36
TVersionInfo.....	37
4.4. Basic API Definition.....	38
ServerManager_Initial.....	39
ServerManager_Release.....	40
ServerManager_GetVersionInfo .....	41
ServerManager_CreateDevice .....	42

ServerManager_DeleteDevice.....	43
ServerManager_SetDeviceProperty .....	44
ServerManager_SetDevicePropertyW.....	45
ServerManager_GetNameList .....	46
ServerManager_GetValueByName .....	47
ServerManager_SetValueByName .....	48
ServerManager_FreeOperator .....	49
ServerManager_SetPTZType .....	50
ServerManager_StopRequest .....	51
4.5. Non-blocking mode API Definition.....	52
ServerManager_GetDIStatus .....	53
ServerManager_GetDOStatus.....	54
ServerManager_GetMotionDetectionInfo .....	55
ServerManager_GetPrivateMaskInfo .....	57
ServerManager_MoveCamera .....	59
ServerManager_OpenDevice .....	61
ServerManager_ReadData.....	62
ServerManager_SendHttpCommand .....	64
ServerManager_SendPTZCommand .....	66
ServerManager_SetDOStatus .....	68
ServerManager_SetMotionDetectionInfo.....	70
ServerManager_SetPrivateMaskInfo.....	72
ServerManager_UartRead.....	74
ServerManager_UartReadAFWrite.....	76
ServerManager_UartWrite.....	78
ServerManager_UpdateLocalConfig .....	80
ServerManager_UpdateRemoteConfig .....	81
ServerManager_AbortProcess .....	82
4.6. Block mode API Definition.....	83
ServerManager_GetDIStatusBlock.....	84
ServerManager_GetDOStatusBlock.....	85
ServerManager_GetMotionDetectionInfoBlock .....	86
ServerManager_GetPrivateMaskInfoBlock.....	87
ServerManager_MoveCameraBlock.....	88
ServerManager_OpenDeviceBlock .....	90
ServerManager_OpenDeviceBlockW .....	91

ServerManager_OpenDeviceLiteBlock.....	92
ServerManager_OpenDeviceLiteBlockW .....	94
ServerManager_ReadDataBlock.....	96
ServerManager_SendHttpCommandBlock.....	97
ServerManager_SendHttpCommandBlockW .....	98
ServerManager_SendHttpCommandReadBlock .....	99
ServerManager_SendHttpCommandReadBlockW.....	101
ServerManager_SendPTZCommandBlock .....	103
ServerManager_SetDOStatusBlock .....	105
ServerManager_SetMotionDetectionInfoBlock.....	106
ServerManager_SetPrivateMaskInfoBlock .....	107
ServerManager_UartReadBlock.....	108
ServerManager_UartReadAFWriteBlock.....	110
ServerManager_UartWriteBlock .....	112
ServerManager_UpdateLocalConfigBlock.....	114
ServerManager_UpdateRemoteConfigBlock.....	115

## 5. Append

5.1. Error Code.....	116
----------------------	-----

# 1. OVERVIEW

## 1.1. Introduction

This document describes how to use the ServerManager module to get/set system parameters or status for your server.

In old type of server, we use the ServerUtil module to do it. But in new type of servers, they have a brand new parameter retrieving system. So we need different way to get/set them and it is the goal of the ServerManager.

Furthermore the communicate way with server is improved. It become more efficient and more resource saving.

The ServerManager is the module to integrate new module and ServerUtil. It wraps two different style of server control interface as one. The users do not need to know which module supports which models. The single and simple interface provides the users an easy way to control the server.

It includes get/set system parameters, get/set DIDO, send PTZ commands and read/write serial port. To use the ServerManager component, user should include "ControlCommon.h" and "ServerManager.h" in their application.

## 1.2. File Structure

FILE	DESCRIPTION
doc\VNDP_ServerManager_API.doc	This manual
lib\vd_ServerManager.lib	The dynamic linking library
lib\NetScheduler.dll	The dynamic runtime library
lib\ServerController.dll	The dynamic runtime library
lib\ServerControllerLoader.dll	The dynamic runtime library
lib\ServerManager.dll	The dynamic runtime library
lib\ServerUtilityLoader.dll	The dynamic runtime library
lib\ServerUtil.dll	The dynamic runtime library
lib\SrvDepResource.dll	The dynamic runtime library
inc\ControlCommon.h	Common definition file
inc\ServerManager.h	ServerManager Header file
inc\ServerManagerError.h	ServerManager error code definition

## 2. PROGRAMMER'S GUIDE

VIVOTEK CONFIDENTIAL  
2009.08.31

## 2.1. Using ServerChannel Module

### Create an Access to the Camera Server

Before controlling the camera server, you must first gain the access. Call `ServerManager_Initial()` to get the `ServerManager` control handle and specify the maximum device number that it can control.

After get the control handle, you can use this handle to create device instance by calling `ServerManager_CreateDevice`. Then, you can set the property to the created device instance by call `ServerManager_SetDeviceProperty()`.

Finally, calling `ServerManager_OpenDevice()` or `ServerManager_OpenDeviceBlock()` to gain the access to the camera server.

After the above, you can start to get/set system parameters, get/set DDO, send PTZ commands and read/write serial port.

### 3. Sample Code

VIVOTEK CONFIDENTIAL  
2009.08.31



## 3.1. Get/Set Network Information

### DESCRIPTION

Create a device and get network information.

### SAMPLE CODE

#### STEP 1. Initial ServerManager and Create a Device

```
// Set ServerManager.dll version
tVerInfo.wMajor = SRVMNGR_VERSION_MAJOR;
tVerInfo.wMinor = SRVMNGR_VERSION_MINOR;
tVerInfo.wBuild = SRVMNGR_VERSION_BUILD;
tVerInfo.wRevision = SRVMNGR_VERSION_REVISION;

// Initial server manager handle
// param dwMaxDevice The maximum number of devices are allowed to open.
// param dwFlag The flag to use in the ServerManager module.
ServerManager_Initial(&hSrvMgr, CONST_MAX_DEVICE, 0, &tVerInfo);
// Create device handle
ServerManager_CreateDevice(hSrvMgr, &hDevice);
```

#### STEP 2. Set Device Property

```
TSRVMNGR_DEV_PROPERTY tDevProperty = {0};
strcpy(tDevProperty.tServerProperty.szHost, "192.168.1.100");
strcpy(tDevProperty.tServerProperty.szUserName, "root");
strcpy(tDevProperty.tServerProperty.szPassword, "123");
tDevProperty.tServerProperty.dwHttpPort = 80;
tDevProperty.tServerProperty.dwTimeout = 30000;

// Update device property
ServerManager_SetDeviceProperty(hDevice, &tDevProperty);
```

### STEP 3. Gain an Access to the Server

```
// Open device (block mode)

// param lpszModelName The char pointer to receive the retrieved model name
// param dwMaxSize     The maximum size of model name
char szModel[MAX_PATH + 1] = {0};
ServerManager_OpenDeviceBlock(hDevice, szModel, MAX_PATH);
```

### STEP 4. Get Network Information, ex. Rtsp Port

```
char szValue[MAX_PATH + 1] = {0};
DWORD dwBufSize = MAX_PATH;

// param pszName      The pointer to the name of the key you want to get the value.
// param pszValue      The pointer to the buffer to receive the value.
// param pdwValueSize  The pointer to the DWORD to indicate the size of buffer.
//                    After the function returned, it will be filled with the length of value.
ServerManager_GetValueByName(hDevice, "network_rtsp_port", szValue, &dwBufSize);
```

### STEP 5. Set Network Information, ex. Rtsp Port

```
// Set Network Information : ex. Set Server Rtsp port
// param pszName      The pointer to the name of the key you want to update the value.
// param pszValue      The pointer to the buffer which contain the value.
// param bMustExist    The key must exist or not.
ServerManager_SetValueByName(hDevice, "network_rtsp_port", "554", TRUE);
```

### STEP 6. Update the Local Change to the Remote Server

```
ServerManager_UpdateRemoteConfigBlock(hDevice);
```

### STEP 7. Delete Device and Release ServerManager

```
ServerManager_DeleteDevice(&hDevice);
ServerManager_Release(&hSrvMgr);
```

## TIPS

ServerManager\_GetValueByName()/ServerManager\_SetValueByName() function is getting/setting the local information that retrieved by ServerManager\_OpenDeviceBlock. If you want to update the local changes to the camera server, you must call ServerManager\_UpdateRemoteConfigBlock() or ServerManager\_UpdateRemoteConfig().

The key name can be retrieved from the server by send HTTP request "/cgi-bin/admin/

getparam.cgi", or you can call ServerManager\_GetNameList() and input "" to the second parameter and it will return all key names.

You can refer to the SDK package for full sample code.

VIVOTEK CONFIDENTIAL  
2009.08.31

## 3.2. Send PTZ Command

### DESCRIPTION

Create a device and send a PTZ command.

### SAMPLE CODE

#### STEP 1. Initial ServerManager and Create a Device

This code is similar to the above

#### STEP 2. Set Device Property

This code is similar to the above

#### STEP 3. Gain an Access to the Server

This code is similar to the above

#### STEP 4. Send a PTZ Command

```
// Send PTZ command to server (blocking)
// Param dwCamera In the multiple camera model, it indicates the order from the right with
// starting as 1. Otherwise it will be ignored.
// Param ePtzCommand The ESRVCTRL_PTZ_COMMAND.
// Param pszExtraCmd The extra information for certain PTZ command.
// Param bWaitRes The boolean value to wait for the server's response or not.
ServerManager_SendPTZCommandBlock(hDevice, 1, eSrvCtrlPTZMoveDown, "", TRUE);
```

#### STEP 5. Delete Device and Release ServerManager

This code is similar to the above

### TIPS

You can refer to the SDK package for full sample code.

### 3.3. Get/Set DO

#### STEP 1. Initial ServerManager and Create a Device

This code is similar to the above

#### STEP 2. Set Device Property

This code is similar to the above

#### STEP 3. Gain an Access to the Server

This code is similar to the above

#### STEP 4. Get DO Information from Server

```
ESrvCtrlDiDoLevel aeDOSStatus[CONST_MAX_DIDO_NUM];
DWORD dwDoNum = CONST_MAX_DIDO_NUM;

// Get DO status
// Param peDOSStatus The pointer to a ESrvCtrlDiDoLevel array.
// Param pdwMaxDONum The pointer to a DWORD to represent the size value of DO
// status array buffer. And the module will replace it with the actual size value
// of DO status array. If the actual size is more than *pdwMaxDINum,
// the module will return error. And the give buffer will not be filled.
ServerManager_GetDOSStatusBlock(hDevice, aeDOSStatus, &dwDoNum);
```

### STEP 5. Change DO array and set to the server.

```
for(iIndex = 0; iIndex < (int) dwDoNum; iIndex++)
{
    if (aeDOStatus[iIndex] == eSrvCtrlDiDoLow)
    {
        // set DO to eSrvCtrlDiDoHigh;
        aeDOStatus[iIndex] = eSrvCtrlDiDoHigh;
    }
    else
    {
        // set DO to eSrvCtrlDiDoLow
        aeDOStatus[iIndex] = eSrvCtrlDiDoLow;
    }
}
// Set DO status
// Param peDOStatus The pointer to a ESrvCtrlDiDoLevel array.
// Param dwMaxDONum The size value of DO status array
// Param bWaitRes The Boolean value to wait for the server's response or not.
ServerManager_SetDOStatusBlock(hDevice, aeDOStatus, dwDoNum, FALSE);
```

### STEP 6. Delete Device and Release ServerManager

This code is similar to the above

### TIPS

You can refer to the SDK package for full sample code.

## 3.4. Send HTTP Command to Server (Non-Blocking Mode)

### STEP 1. Prepare the callback functions

```
SCODE __stdcall ProcNotify(TSVRCTRL_NOTIFY_CONT *ptContext)
```

### STEP 2. Initial ServerManager and Create a Device

This code is similar to the above

### STEP 3. Set Device Property

This code is similar to the above

### STEP 4. Gain an Access to the Server (Non-Blocking)

```
// Non-block setting, Callback information
tCBContent.hEvent = hWaitEvent;
tNotify.pvContext = &tCBContent;
tNotify.pfCB = ProcNotify;
// Open device (non-block mode)
ServerManager_OpenDevice(hDevice, &tNotify);
```

### STEP 5. Send HTTP Command

```
// Non-block setting, Callback information
tCBContent.hEvent = hWaitEvent;
tNotify.pvContext = &tCBContent;
tNotify.pfCB = ProcNotify;
// STEP 7: Send HTTP command (non-block) to server
// Param pszCommand The string pointer to an http command.
// Param bPost True is for POST, false for GET
// Param ptNotify The pointer TSVRCTRL_NOTIFY contains the callback function and
// related information. It cannot be null and the content should be valid.
// Param phProc The pointer to a http operation handle. The http operation handle
// is used to read more data. The module will pass a handle to the caller
// if there is more data to be read. If none, the phOper will be null.
// The users do not need to free the http operation handle.
ServerManager_SendHttpCommand(hDevice, "/cgi-bin/admin/setparam.cgi?system_
```

hostname="Network Camera", TRUE, &tNotify, NULL);

## **TIPS**

The ServerManager\_OpenDevice() function and ServerManager\_SendHttpCommand() function are asynchronous functions, the result should be get in SVRCTRL\_STATUS\_CALLBACK call back function.

You can refer to the SDK package for full sample code.

VIVOTEK CONFIDENTIAL  
2009.08.31



## 4. API Reference

This chapter contains the API function calls for the ServerManager.

VIVOTEK CONFIDENTIAL  
2009.08.31

## 4.1. Enumeration

The enumeration used is depicted here.

VIVOTEK CONFIDENTIAL  
2009.08.31

## EJOYSTICK\_ZOOMING\_COMMAND

This enumeration indicates the simulation of joystick zooming commands.

```
typedef enum
{
    eJoystickZoomStop,
    eJoystickZoomWide,
    eJoystickZoomTele,
} EJOYSTICK_ZOOMING_COMMAND;
```

### Values

#### **eJoystickZoomStop**

Stop the zooming operation.

#### **eJoystickZoomWide**

Start zooming wide.

#### **eJoystickZoomTele**

Start zooming tele.

### Remarks

### Requirements

ControlCommon.h

## ESrvCtrlDiDoLevel

This enumeration indicates the DIDO status

```
typedef enum
{
    eSrvCtrlDiDoNone,
    eSrvCtrlDiDoLow,
    eSrvCtrlDiDoHigh
} ESrvCtrlDiDoLevel;
```

### Values

#### **eSrvCtrlDiDoNone**

This type means the server does not have digital output

#### **eSrvCtrlDiDoLow**

This type means the digital output status of server is at low level

#### **eSrvCtrlDiDoHigh**

This type means the digital output status of server is at high level

### Remarks

### Requirements

ControlCommon.h

## ESrvCtrlPTZType

This enumeration indicates the PTZ type.

```
typedef enum
{
    eSrvCtrlPPTZ,
    eSrvCtrlEPTZ
} ESrvCtrlPTZType;
```

### Values

#### **eSrvCtrlPPTZ**

Use physical PTZ command.

#### **eSrvCtrlEPTZ**

Use digital PTZ command.

### Remarks

Make sure the camera server support digital PTZ command if you want to control the camera by digital PTZ commands.

### Requirements

ControlCommon.h

## ESRVCTRL\_PTZ\_COMMAND

This enumeration indicates the PTZ commands

```
typedef enum
{
    eSrvCtrlPTZMoveUp,
    eSrvCtrlPTZMoveDown ,
    eSrvCtrlPTZMoveLeft,
    eSrvCtrlPTZMoveRight,
    eSrvCtrlPTZMoveHome,
    eSrvCtrlPTZZoomIn,
    eSrvCtrlPTZZoomOut,
    eSrvCtrlPTZFocusNear,
    eSrvCtrlPTZFocusFar,
    eSrvCtrlPTZFocusAuto,
    eSrvCtrlPTZPresetAdd,
    eSrvCtrlPTZPresetDel,
    eSrvCtrlPTZPresetRecall,
    eSrvCtrlPTZPanSpeed,
    eSrvCtrlPTZTiltSpeed,
    eSrvCtrlPTZStartPan,
    eSrvCtrlPTZStopPan,
    eSrvCtrlPTZStartPatrol,
    eSrvCtrlPTZIrisOpen,
    eSrvCtrlPTZIrisClose,
    eSrvCtrlPTZIrisAuto,
    eSrvCtrlPTZZoomSpeed,
    eSrvCtrlPTZSetHome,
    eSrvCtrlPTZDefaultHome,
    eSrvCtrlPTZFocusSpeed,
    eSrvCtrlPTZCustom
} ESrvCtrlIDiDoLevel;
```

### Values

#### **eSrvCtrlPTZMoveUp**

Move the camera one step up. The step is decided on tilt speed.

**eSrvCtrlPTZMoveDown**

Move the camera one step down. The step is decided on tilt speed.

**eSrvCtrlPTZMoveLeft**

Move the camera one step left. The step is decided on pan speed.

**eSrvCtrlPTZMoveRight**

Move the camera one step right. The step is decided on pan speed.

**eSrvCtrlPTZMoveHome**

Move the camera back to its home position. Usually, this is the position when camera boots up.

**eSrvCtrlPTZZoomIn**

Zoom in the camera to see more details. Note that this command only works for camera that supports zooming. The return code will be OK even if the camera does not support zooming. The OK means the command is sent to server successfully.

**eSrvCtrlPTZZoomOut**

Zoom out the camera to see wider view. Note that this command only works for camera that supports zooming. The return code will be OK even if the camera does not support zooming. The OK means the command is sent to server successfully.

**eSrvCtrlPTZFocusNear**

Set the focus of the camera to shorter focal distance. Note that this command only works for camera that supports focus adjustment. The return code will be OK even if the camera does not support focus adjustment. The OK means the command is sent to server successfully.

**eSrvCtrlPTZFocusFar**

Set the focus of the camera to longer focal distance. Note that this command only works for camera that supports focus adjustment. The return code will be OK even if the camera does not support focus adjustment. The OK means the command is sent to server successfully.

**eSrvCtrlPTZFocusAuto**

Let the camera adjust focal distance automatically. After this command is set, every time when camera move, the camera will try to locate the proper focal distance. Note that this command only works for camera that supports focus adjustment. The return code will be OK even if the camera does not support focus adjustment. The OK means the command is sent to server successfully.

**eSrvCtrlPTZPresetAdd**

Add a new preset location. The preset location is a string that indicates a saved coordinate in camera.

**eSrvCtrlPTZPresetDel**

Remove a preset location in camera.

**eSrvCtrlPTZPresetRecall**

Move the camera to a preset location

### **eSrvCtrlPTZPanSpeed**

Adjust the camera's pan speed.

### **eSrvCtrlPTZTiltSpeed**

Adjust the camera's tilt speed.

### **eSrvCtrlPTZStartPan**

Let PT or PTZ camera (both are built in IP camera) to start pan for a round.

### **eSrvCtrlPTZStopPan**

Stop the auto pan or patrol in PT or PTZ camera (both are built in IP camera).

### **eSrvCtrlPTZStartPatrol**

Let PT or PTZ camera (both are built in IP camera) to patrol the predefined route. User must setup the patrol route before this command takes effect.

### **eSrvCtrlPTZIrisOpen**

Increase the IRIS level.

### **eSrvCtrlPTZIrisClose**

Decrease the IRIS level.

### **eSrvCtrlPTZIrisAuto**

Set the IRIS to auto mode

### **eSrvCtrlPTZZoomSpeed**

Adjust the camera's zoom speed

### **eSrvCtrlPTZSetHome**

Set the new home position

### **eSrvCtrlPTZDefaultHome**

Set back to default home position

### **eSrvCtrlPTZFocusSpeed**

Adjust the camera's focus speed.

### **eSrvCtrlPTZCustom**

Call custom command. This is useful for video server that might connect to analog camera that provides more commands except the standard pan and tilt command.

## **Remarks**

## **Requirements**

ControlCommon.h



## 4.2. Callback Function

The Callback function is depicted here.

VIVOTEK CONFIDENTIAL  
2009.08.31

## SVRCTRL\_STATUS\_CALLBACK

This status callback is used when non-blocking mode is specified. Non-blocking mode let application to submit a work and go ahead to handle other task. This module will call application back when work done through this callback.

### Syntax

```
typedef SCODE  
(__stdcall *SVRCTRL_STATUS_CALLBACK) (    TSVRCTRL_NOTIFY_CONT    );  
                                         *ptContent
```

### Parameters

#### ptContent

[in] it is the pointer to TSVRCTRL\_NOTIFY\_CONT that contain the callback information.

### Return Values

Ignored

### Remarks

### Requirements

ControlCommon.h

## 4.3. Data Structure

The data structure is depicted here.

VIVOTEK CONFIDENTIAL  
2009.08.31

# TSRVCTRL\_DEV\_PROPERTY

This structure indicates the properties of device. When opening the device, you must set the properties for the device.

```
typedef struct
{
    char            szHost [SRVCTRL_MAX_HOST_LEN+1];
    char            szUserName [SRVCTRL_MAX_USERNAME_LEN+1];
    char            szPassword [SRVCTRL_MAX_USERPASS_LEN+1];
    DWORD           dwFtpPort;
    DWORD           dwHttpPort;
    DWORD           dwTimeout;
    BOOL            bUseSSL;
}TSRVCTRL_DEV_PROPERTY, *PTSRVCTRL_DEV_PROPERTY;
```

## Members

### **szHost**

IP address

### **szUserName**

User name

### **szPassword**

Password

### **dwHttpPort**

Http port for the server

### **dwFtpPort**

Ftp port for the server

### **dwTimeout**

This is the timeout for the operation in milliseconds.

### **bUseSSL**

Connect to server by using SSL protocol

## Remarks

## Requirements

ControlCommon.h

## TSRVCTRL\_DEV\_PROPERTYW

This structure indicates the properties of device. When opening the device, you must set the properties for the device. This is wide-character version of TSRVCTRL\_DEV\_PROPERTY.

```
typedef struct
{
    wchar_t          szHost [SRVCTRL_MAX_HOST_LEN+1] t;
    wchar_t          szUserName [SRVCTRL_MAX_USERNAME_
    wchar_t          LEN+1];
    DWORD            szPassword [SRVCTRL_MAX_USERPASS_LEN+1];
    DWORD            dwHttpPort;
    DWORD            dwFtpPort;
    BOOL             dwTimeout;
                    bUseSSL;
} TSRVCTRL_DEV_PROPERTYW, *PTRSVCTRL_DEV_PROPERTYW;
```

### Members

**szHost**

IP address

**szUserName**

User name

**szPassword**

Password

**dwHttpPort**

Http port for the server

**dwFtpPort**

Ftp port for the server

**dwTimeout**

This is the timeout for the operation in milliseconds.

**bUseSSL**

Connect to server by using SSL protocol

### Remarks

### Requirements

ControlCommon.h

# TSRVCTRL\_MODETECT\_INFO

This structure contains the information of one motion detection window.

```
typedef struct
{
    TCHAR                                szName[SRVCTRL_MAX_
    int                                  WINDOWNAME+1];
    int                                  nX;
    int                                  nY;
    int                                  nW;
    int                                  nH;
    int                                  nPercent;
    int                                  nSensitivity;
    BOOL                                 bWindowEnabled;
} TSRVCTRL_MODETECT_INFO;
```

## Members

### szName

This is the name for the motion detection window.

### nX

Specify the x coordinate for the window related to the upper left corner of the video. The value is for normal size image. If the image is shown in other size, translation is needed.

### nY

Specify the x coordinate for the window related to the upper left corner of the video. The value is for normal size image. If the image is shown in other size, translation is needed.

### nW

Specify the width for the window. The value is for normal size image. If the image is shown in other size, translation is needed.

### nH

Specify the height for the window. The value is for normal size image. If the image is shown in other size, translation is needed.

### nPercent

Specify the percentage of the pixels in the specified window that the difference of two adjacent frames is larger than the threshold (as specified in nSensitivity) to be judged as triggered for motion detection.

### nSensitivity

Specify the threshold that one pixel in the window is called "moved" if the difference of two adjacent frame of the same pixel is larger than this value.

**bWindowEnabled**

Specify the window is enabled or not.

**Remarks****Requirements**

ControlCommon.h

VIVOTEK CONFIDENTIAL  
2009.08.31

# TSRVCTRL\_PRIVATEMASK\_INFO

This structure contains the information of one private mask window.

```
typedef struct
{
    TCHAR                szName[SRVCTRL_MAX_
                          WINDOWNAME+1];
    int                  nX;
    int                  nY;
    int                  nW;
    int                  nH;
    BOOL                 bWindowEnabled;
} TSRVCTRL_PRIVATEMASK_INFO;
```

## Members

### **szName**

This is the name for the private mask window.

### **nX**

Specify the x coordinate for the window related to the upper left corner of the video. The value is for normal size image. If the image is shown in other size, translation is needed.

### **nY**

Specify the x coordinate for the window related to the upper left corner of the video. The value is for normal size image. If the image is shown in other size, translation is needed. .

### **nW**

Specify the width for the window. The value is for normal size image. If the image is shown in other size, translation is needed.

### **nH**

Specify the height for the window. The value is for normal size image. If the image is shown in other size, translation is needed.

### **bWindowEnabled**

Specify the window is enabled or not.

## Remarks

## Requirements

ControlCommon.h



## TSVRCTRL\_NOTIFY

This structure is used to specify the notification interface and parameter for asynchronous operation.

```
typedef struct
{
    SVRCTRL_STATUS_CALLBACK      pfCB;
    void *                       pvContext;
} TSVRCTRL_NOTIFY;
```

### Members

#### **pfCB**

the [SVRCTRL\\_STATUS\\_CALLBACK](#) that specify the callback function to be used to notify.

#### **pvContext**

Specify the context value that is to be sent back when callback function gets called.

### Remarks

### Requirements

ControlCommon.h

# TSVRCTRL\_NOTIFY\_CONT

The structure is used to hold the notified data when callback is called.

```
typedef struct
{
    HANDLE                                pfCB;
    void *                                pvContext;
    SCODE                                 scStatusCode;
    void *                                pvParam1;
    void *                                pvParam2;
} TSVRCTRL_NOTIFY_CONT;
```

## Members

### pfCB

Specify the callback function to be used to notify.

### pvContext

Specify the context value that is to be sent back when callback function gets called.

### scStatusCode

Specify the context value that is to be sent back when callback function gets called.  
pvParam1

**Specify the context value that is to be sent back when callback function gets called. pvParam2**

Specify the context value that is to be sent back when callback function gets called.

## Remarks

## Requirements

ControlCommon.h

## TSRVMNGR\_DEV\_PROPERTY

The structure is used to hold the property of target host and the priority interface to connect the server.

```
typedef struct
{
    TSRVCTRL_DEV_PROPERTY    tServerProperty;
    BOOL                     bServerUtlPriority;
} TSRVMNGR_DEV_PROPERTY;
```

### Members

#### **tServerProperty;**

The [TSRVCTRL\\_DEV\\_PROPERTY](#) contain the property of target host.

#### **bServerUtlPriority;**

It indicates that we first use ServerUtl module or not. We will use ServerUtl module to connect first, if fail, we will automatically roll to ServerController to do it. If the user does not know which module to use first, just set it false.

### Remarks

### Requirements

ServerManager.h

## TSRVMNGR\_DEV\_PROPERTYW

The structure is used to hold the property of target host and the priority interface to connect the server. This is wide-character version of TSRVMNGR\_DEV\_PROPERTY.

```
typedef struct
{
    TSRVCTRL_DEV_PROPERTYW    tServerProperty;
    BOOL                      bServerUtilPriority;
} TSRVMNGR_DEV_PROPERTYW;
```

### Members

#### **tServerProperty;**

The [TSRVCTRL\\_DEV\\_PROPERTYW](#) contain the property of target host.

#### **bServerUtilPriority;**

It indicates that we first use ServerUtl module or not. We will use ServerUtl module to connect first, if fail, we will automatically roll to ServerController to do it. If the user does not know which module to use first, just set it false.

### Remarks

### Requirements

ServerManager.h

## TVersionInfo

This structure contains version information.

```
typedef struct
{
    WORD                wMajor;
    WORD                wMinor;
    WORD                wBuild;
    WORD                wRevision;
} TVersionInfo;
```

### Members

#### **wMajor**

Major version number.

#### **wMinor**

Minor version number

#### **wBuild**

The build number.

#### **wRevision**

The revision number.

### Remarks

### Requirements

ControlCommon.h

## 4.4. Basic API Definition

The API definition is depicted here.

VIVOTEK CONFIDENTIAL  
2009.08.31

# ServerManager\_Initial

Create a ServerManager object instance and initialize it.

## Syntax

```
SCODE ServerManager__Initial (    HANDLE *phObject
                                DWORD  dwMaxDevice
                                DWORD  dwFlag
                                TVersionInfo* ptVersion    );
```

## Parameters

### **\*phObject**

[out] the pointer to a handle to receive the ServerManager object instance.

### **dwMaxDevice**

[in] the maximum number of devices are allowed to open

### **dwFlag**

[in] the flag to use in the ServerManager module

### **ptVersion**

[in] the pointer to a [TVersionInfo](#) contain the version of the ServerManager module

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

## Requirements

ServerManager.h

## See Also

[ServerManager\\_Release](#)

## ServerManager\_Release

Delete a ServerManager object instance. This function must be called to prevent the resource leaking whenever the ServerManager handle is no longer used.

### Syntax

<b>SCODE</b> ServerManager_Release ( <b>HANDLE</b> *phObject );
---

### Parameters

#### phObject

[in out] the pointer to the handle of the ServerManager object instance be released. It returned from [ServerManager\\_Initial](#).

### Return Values

Returns S\_OK if successful, or an error value otherwise.

### Remarks

### Requirements

ServerManager.h

### See Also

[ServerManager\\_Initial](#)



## ServerManager\_GetVersionInfo

This function is used to retrieve the version information of the current module.

### Syntax

<b>SCODE</b> ServerManager_GetVersionInfo ( <b>TVersionInfo</b> *ptVersion    );
--

### Parameters

**ptVersion**

[in out] the pointer to the struct of the module's version information.

### Return Values

Returns S\_OK if successful, or an error value otherwise.

### Remarks

### Requirements

ServerManager.h

### See Also

[ServerManager\\_Initial](#)

## ServerManager\_CreateDevice

Create a device object instance. The device is used to control a server. We must create a device handle first. And we can manage the device base on this handle

### Syntax

<b>SCODE</b> ServerManager_CreateDevice ( <b>HANDLE</b> hObject, <b>HANDLE</b> *phDevice,     );
---

### Parameters

#### **hObject**

[in] the handle of the ServerManager object instance. It returned from [ServerManager\\_Initial](#).

#### **phDevice**

[out] the pointer to a handle to receive the created device object instance.

### Return Values

Returns S\_OK if successful, or an error value otherwise.

### Remarks

.

### Requirements

ServerManager.h

### See Also

[ServerManager\\_Initial](#) [ServerManager\\_DeleteDevice](#)

# ServerManager\_DeleteDevice

Delete the specified device object instance

## Syntax

<b>SCODE</b> ServerManager_DeleteDevice ( <b>HANDLE</b> *phDevice );
--

## Parameters

**phDevice**

[in out] the pointer to the handle of the device object. It returned from [ServerManager\\_CreateDevice](#)

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#)

# ServerManager\_SetDeviceProperty

Set the property of the specified device object.

## Syntax

```
SCODE ServerManager_SetDeviceProperty ( HANDLE hDevice  
                                         TSRVMNGL_DEV_PROERTY  
                                         *ptDevProperty  
                                         );
```

## Parameters

### hDevice

[in] the handle of the device object. It returned from [ServerManager\\_CreateDevice](#).

### ptDevProperty

[in] the pointer to [TSRVMNGL\\_DEV\\_PROERTY](#) to store to store the property of target host

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

If you want to setup a used device handle, the module may call [ServerManager\\_StopRequest](#) to stop the unfinished or pending request. And it will set bBlocking as true to avoid unexpected time delay response problem. So it may take times to wait for the function return and then it could set up the device property.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_StopRequest](#)

# ServerManager\_SetDevicePropertyW

Set the property of the specified device object. This is wide-character version of ServerManager\_SetDeviceProperty.

## Syntax

<b>SCODE</b> <b>ServerManager_SetDevicePropertyW (</b>	<b>HANDLE</b> hDevice <b>TSRVMNGL_DEV_PROERTYW</b> <b>*ptDevPropertyW</b> <b>);</b>
---	--

## Parameters

### hDevice

[in] the handle of the device object. It returned from [ServerManager\\_CreateDevice](#).

### ptDevPropertyW

[in] the pointer to [TSRVMNGL\\_DEV\\_PROERTYW](#) to store to store the property of target host

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

If you want to setup a used device handle, the module may call [ServerManager\\_StopRequest](#) to stop the unfinished or pending request. And it will set bBlocking as true to avoid unexpected time delay response problem. So it may take times to wait for the function return and then it could set up the device property.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_SetDeviceProperty](#)

## ServerManager\_GetNameList

Retrieve the value entries under the specified name.

Only the first layer names under root key are retrieved. The name will be cascaded with root name to get full name. If the buffer size is not enough error will be returned, and \*pdwValueSize will give the idea how large the buffer is needed

### Syntax

<b>SCODE</b> ServerManager_GetNameList (	<b>HANDLE</b> hDevice,
	<b>const TCHAR*</b> pszName,
	<b>TCHAR*</b> pszNameList,
	<b>DWORD*</b> pdwValueSize
	<b>);</b>

### Parameters

#### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#)

#### pszName

[in] the pointer to the name of the key you want to get the value.

#### pszNameList

[out] the pointer to the buffer to receive the value.

#### pdwValueSize

[in out] the pointer to the DWORD to indicate the size of buffer. After the function returned, it will be filled with the length of value.

### Return Values

Returns S\_OK if successful, or an error value otherwise.

### Remarks

In ServerUtl applied case, this function may not find the appropriate one to do it. The module will return ERR\_NOT\_IMPLEMENT.

### Requirements

ServerManager.h

### See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)

# ServerManager\_GetValueByName

Get a value from local buffer.

## Syntax

```
SCODE ServerManager_GetValueByName ( HANDLE hDevice,  
                                     const TCHAR* pszName,  
                                     TCHAR* pszValue,  
                                     DWORD* pdwValueSize );
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#)

### pszName

[in] the pointer to the name of the key you want to get the value.

### pszValue

[out] the pointer to the buffer to receive the value.

### pdwValueSize

[in out] the pointer to the DWORD to indicate the size of buffer. After the function returned, it will be filled with the length of value.

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)

# ServerManager\_SetValueByName

Set a value to local buffer.

## Syntax

```
SCODE ServerManager_SetValueByName ( HANDLE hDevice,  
                                     const TCHAR* pszName,  
                                     const TCHAR* pszValue,  
                                     BOOL bMustExist );
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### pszName

[in] the pointer to the name of the key you want to update the value.

### pszValue

[in] the pointer to the buffer which contain the value.

### bMustExist

[in] the key must exist or not.

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)



# ServerManager\_FreeOperator

Return the operator to hDevice.

## Syntax

```
SCODE ServerManager_FreeOperator ( HANDLE hOper );
```

## Parameters

**hOper**

[in] the http operation handle. It returned from [ServerController\\_SendHttpCommand](#) or [ServerController\\_SendHttpCommandReadBlock](#)

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

## Requirements

ServerManager.h

## See Also

[ServerController\\_SendHttpCommand](#) [ServerController\\_SendHttpCommandReadBlock](#)

# ServerManager\_SetPTZType

Set PTZ command type, physical or digital.

## Syntax

```
SCODE ServerManager_SetPTZType ( HANDLE hDevice  
                                ESrvCtrlPTZType ePTZType );
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### ePTZType

[in] the type of PTZ command send by ServerManager\_SendPTZCommand, ServerManager\_MoveCamera, ServerManager\_SendPTZCommandBlock, ServerManager\_MoveCameraBlock.

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

## Requirements

ServerManager.h

## See Also

ServerManager\_SendPTZCommand, ServerManager\_MoveCamera, ServerManager\_SendPTZCommandBlock, ServerManager\_MoveCameraBlock.

## ServerManager\_StopRequest

This function is used to stop all pending or unfinished requests for a device handle. If you are calling this before calling [ServerManager\\_DeleteDevice](#), set the bBlocking flag to FALSE.

### Syntax

<b>SCODE</b> ServerManager_StopRequest ( <b>HANDLE</b> hDevice, <b>BOOL</b> bBlocking           );
---

### Parameters

hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

**bBlocking**

[in] the boolean value to stop request in block mode. If yes, it will block until all connections done. You better set block mode when you want to reuse this device to operate another different host

### Return Values

Returns S\_OK if successful, all request are stopped successfully.

### Remarks

### Requirements

ServerManager.h

### See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_DeleteDevice](#)

## 4.5. Non-blocking mode API Definition

The non-blocking API definition is depicted here.

The non-block mode functions are all asynchronous. Calling any non-block function will not block the process and it will notify the user with the callback function after the operation is done.

Every non-block function has a parameter which type is pointer of [TSVRCTRL\\_NOTIFY](#). It contains the information of the callback function and user's context value. The user can install the callback function and user's data while call non-block function. Once the operation has been done and the callback function is raised, it will pass that value to the user to specify whatever data user wants.

Non-block function also has a parameter which type is pointer of handle. It is used to receive the instance of the http operation process. The user can control the process base on this handle. If does not use this operation before the callback function is notified, the user can call `ServerManager_AbortProcess` to abort the operation.

# ServerManager\_GetDIStatus

Get current DI information of the server.

## Syntax

```
SCODE ServerManager_GetDIStatus ( HANDLE hDevice,  
                                  TSVRCTRL_NOTIFY * ptNotify,  
                                  HANDLE * phProc );
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### ptNotify

[in] the pointer [TSVRCTRL\\_NOTIFY](#) contains the callback function and related information. It cannot be null and the content should be valid.

### phProc

[out] the pointer to a http operation handle. The http operation handle is used to read more data. The module will pass a handle to the caller if there is more data to be read. If none, the phOper will be null. The users do not need to free the http operation handle. It is kept internally

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will return soon. And callback function will be raised when the operation is really done. In the successful callback function, the pvParam1 contains the number of DI the server has. And pvParam2 would be the ESrvCtrlDiDoLevel array that contains each DI status. The array is invalid after callback returns, so the caller must copy the value if he needs that value later

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#) [ServerManager\\_AbortProcess](#)

# ServerManager\_GetDOStatus

Get current DO information of the server.

## Syntax

```
SCODE ServerManager_GetDOStatus (    HANDLE hDevice,  
                                     TSVRCTRL_NOTIFY * ptNotify,  
                                     HANDLE * phProc  
                                     );
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### ptNotify

[in] the pointer [TSVRCTRL\\_NOTIFY](#) contains the callback function and related information. It cannot be null and the content should be valid.

### phProc

[out] the pointer to a http operation handle. The http operation handle is used to read more data. The module will pass a handle to the caller if there is more data to be read. If none, the phOper will be null. The users do not need to free the http operation handle. It is kept internally.

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will return soon. And callback function will be raised when the operation is really done. In the success notification the pvParam1 contains the number of DO the server has. And pvParam2 would be the ESrvCtrlDiDoLevel array that contains each DO status. The array is invalid after callback returns, so the caller must copy the value if he needs that value later.

In ServerUtl applied case, this function may not find the appropriate one to do it. The module will return ERR\_NOT\_IMPLEMENT.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#) [ServerManager\\_AbortProcess](#)

# ServerManager\_GetMotionDetectionInfo

Get current motion detection setting information of the server.

## Syntax

<b>SCODE</b> <b>ServerManager_GetMotionDetectionInfo (</b>	<b>HANDLE</b> hDevice, <b>DWORD</b> dwCamera, <b>TSVRCTRL_NOTIFY</b> * ptNotify, <b>HANDLE</b> * phProc <b>);</b>
---	---

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### dwCamera

[in] in the multiple camera model, it indicate the order from the right with starting as 1. Otherwise the module will ignore it.

### ptNotify

[in] the pointer [TSVRCTRL\\_NOTIFY](#) contains the callback function and related information. It cannot be null and the content should be valid.

### phProc

[out] the pointer to a http operation handle. The http operation handle is used to read more data. The module will pass a handle to the caller if there is more data to be read. If none, the phOper will be null. The users do not need to free the http operation handle. It is kept internally

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will return soon. And callback function will be raised when the operation is really done. In the success notification, pvParam1 is the TSVRCTRL\_MODETECT\_INFO array that contains motion window information. The array is invalid after callback returns, so the caller must copy the value if he needs that value later.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#) [ServerManager\\_AbortProcess](#)

VIVOTEK CONFIDENTIAL  
2009.08.31



# ServerManager\_GetPrivateMaskInfo

Get the current private mask setting information of the server.

## Syntax

```
SCODE ServerManager_GetPrivateMaskInfo ( HANDLE hDevice,  
                                         DWORD dwCamera,  
                                         TSVRCTRL_NOTIFY * ptNotify,  
                                         HANDLE * phProc );
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### dwCamera

[in] in the multiple camera model, it indicate the order from the right with starting as 1. Otherwise the module will ignore it.

### ptNotify

[in] the pointer [TSVRCTRL\\_NOTIFY](#) contains the callback function and related information. It cannot be null and the content should be valid.

### phProc

[out] the pointer to a http operation handle. The http operation handle is used to read more data. The module will pass a handle to the caller if there is more data to be read. If none, the phOper will be null. The users do not need to free the http operation handle. It is kept internally

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will return soon. And callback function will be raised when the operation is really done. In the success notification, pvParam1 is the [TSRVCTRL\\_PRIVATEMASK\\_INFO](#) array that contains private mask window information. The array is invalid after callback returns, so the caller must copy the value if he needs that value later.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#) [ServerManager\\_AbortProcess](#)

VIVOTEK CONFIDENTIAL  
2009.08.31

# ServerManager\_MoveCamera

Move the camera by the specified coordinate.

## Syntax

```
SCODE ServerManager_MoveCamera (    HANDLE hDevice,  
                                     DWORD dwCamera,  
                                     DWORD dwStream,  
                                     DWORD dwX,  
                                     DWORD dwY,  
                                     DWORD dwDisplayWidth,  
                                     DWORD dwDisplayHeight,  
                                     TSVRCTRL_NOTIFY * ptNotify,  
                                     HANDLE * phProc    );
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### dwCamera

[in] in the multiple camera model, it indicates the order from the right with starting as 1. Otherwise it will be ignored.

### dwStream

[in] in the dual-stream camera model, it indicates resolution of which stream will be get for computing the target position.

### dwX

[in] the x coordinate for the new position. This position is related to the left-upper corner of current camera view. The camera will treat this value as the new center x coordinate. This value should be the normal size of camera video size

### dwY

[in] the y coordinate for the new position. This position is related to the left-upper corner of current camera view. The camera will treat this value as the new center y coordinate. This value should be the normal size of camera video size.

### dwDisplayWidth

[in] The weight of the display video size.

### dwDisplayHeight

[in] The height of the display video size

### ptNotify

[in] the pointer [TSVRCTRL\\_NOTIFY](#) contains the callback function and related information. It cannot be null and the content should be valid.

### **phProc**

[out] the pointer to a http operation handle. The http operation handle is used to read more data. The module will pass a handle to the caller if there is more data to be read. If none, the phOper will be null. The users do not need to free the http operation handle. It is kept internally

### **Return Values**

Returns S\_OK if successful, or an error value otherwise.

### **Remarks**

The function will return soon. And callback function will be raised when the operation is really done.

For supporting digital move command, you must set the device to EPTZ mode by call the ServerManager\_SetPTZType.

### **Requirements**

ServerManager.h

### **See Also**

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#) [ServerManager\\_AbortProcess](#)

# ServerManager\_OpenDevice

Open the specified device to access some information. It will connect to the server and retrieve the server model.

## Syntax

```
SCODE ServerManager_OpenDevice (    HANDLE hDevice,  
                                   TSVRCTRL_NOTIFY * ptNotify  
                                   );
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### ptNotify

[in] the pointer [TSVRCTRL\\_NOTIFY](#) contains the callback function and related information. It cannot be null and the content should be valid.

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

If you want to control the device or get some information from the device, you must open the device first.

The function will return soon. And callback function will be raised when the operation is really done. In the success notification, the pvParam1 contains the string for the model value. User could also use the [ServerManager\\_GetValueByName](#) to get the server model value

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#)

## ServerManager\_ReadData

This function is used to read the data from the http connection. User must ensure that the connection is already connected. Or error will be returned.

### Syntax

```
SCODE ServerManager_ReadData ( HANDLE hOper,  
                               BYTE *pbyBuffer,  
                               DWORD *pdwBufLen,  
                               TSVRCTRL_NOTIFY * ptNotify );
```

### Parameters

#### hOper

[in] the http operation handle. It returned from [ServerController\\_SendHttpCommand](#) or [ServerController\\_SendHttpCommandReadBlock](#)

#### pbyBuffer

[out] the pointer to buffer to receive the http response data

#### pdwBufLen

[in out] the pointer to a DWORD to represent the length of the buffer. After the function return, it will be replaced with the actual length value

#### ptNotify

[in] the pointer [TSVRCTRL\\_NOTIFY](#) contains the callback function and related information. It cannot be null and the content should be valid.

#### phProc

[out] the pointer to a http operation handle. The http operation handle is used to read more data. The module will pass a handle to the caller if there is more data to be read. If none, the phProc will be null. The users do not need to free the http operation handle. It is kept internally

### Return Values

Returns S\_OK if successful, or an error value otherwise.

### Remarks

In ServerUtl applied case, this function may not find the appropriate one to do it. The module will return ERR\_NOT\_IMPLEMENT.

### Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#) [ServerManager\\_AbortProcess](#)

VIVOTEK CONFIDENTIAL  
2009.08.31

## ServerManager\_SendHttpCommand

This function is used to send an http command to server. For POST, the parameter is also put after the question mark as GET does. The result of the command is ignored by this function.

If need to read response data, please use `ServerController_SendHttpCommandReadBlock`.

### Syntax

```
SCODE ServerManager_SendHttpCommand( HANDLE hDevice,
                                     const TCHAR *pszCommand,
                                     BOOL bPost,
                                     TSVRCTRL_NOTIFY * ptNotify,
                                     HANDLE * phOper );
```

### Parameters

#### **hDevice**

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

#### **pszCommand**

[in] the string pointer to an http command.

#### **bPost**

[in] Is the command sends by POST or GET? True is for POST, false for GET.

#### **ptNotify**

[in] the pointer [TSVRCTRL\\_NOTIFY](#) contains the callback function and related information. It cannot be null and the content should be valid.

#### **phOper**

[out] the pointer to a http operation handle. The http operation handle is used to read more data. The module will pass a handle to the caller if there is more data to be read. If none, the phOper will be null. The users do not need to free the http operation handle. It is kept internally

### Return Values

Returns `S_OK` if successful, or an error value otherwise.

### Remarks

The function will return soon. And callback function will be raised when the operation is really done.



## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#) [ServerManager\\_AbortProcess](#)

VIVOTEK CONFIDENTIAL  
2009.08.31

# ServerManager\_SendPTZCommand

Send the PTZ command to the server.

## Syntax

```
SCODE ServerManager_SendPTZCommand (  
  
    HANDLE hDevice,  
    DWORD dwCamera,  
    ESRVCTRL_PTZ_COMMAND  
    ePtzCommand,  
    const TCHAR * pszExtraCmd,  
    TSVRCTRL_NOTIFY * ptNotify,  
    HANDLE * phProc  
);
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### dwCamera

[in] in the multiple camera model, it indicates the order from the right with starting as 1. Otherwise it will be ignored.

### ePtzCommand

[in] the [ESRVCTRL\\_PTZ\\_COMMAND](#) to represent a command will be sent to the target device. Not all commands are accepted in every camera, so you may check the user's manual to make sure it

### pszExtraCmd

[in] the extra information for certain PTZ command. Here is a list for the command

- eSrvCtrlPTZPresetAdd: the pointer to the name of the new preset location
- eSrvCtrlPTZPresetDel: the pointer to the name of preset location to be deleted
- eSrvCtrlPTZPresetRecall: the pointer to the name of the preset location to be recalled
- eSrvCtrlPTZPanSpeed: the pointer to the pan speed in string (number presented in string format)
- eSrvCtrlPTZTiltSpeed: the pointer to the tilt speed in string (number presented in string format)
- eSrvCtrlPTZFocusSpeed: the pointer to the tilt speed in string (number presented in string format)

- eSrvCtrlPTZCustom: the pointer to the customized command ID in string  
(number presented in string format)

**ptNotify**

[in] the pointer [TSVRCTRL\\_NOTIFY](#) contains the callback function and related information. It cannot be null and the content should be valid.

**phProc**

[out] the pointer to a http operation handle. The http operation handle is used to read more data. The module will pass a handle to the caller if there is more data to be read. If none, the phProc will be null. The users do not need to free the http operation handle. It is kept internally

**Return Values**

Returns S\_OK if successful, or an error value otherwise.

**Remarks**

The function will return soon. And callback function will be raised when the operation is really done.

For supporting digital PTZ command, you must set the device to EPTZ mode by call the `ServerManager_SetPTZType`, otherwise the device will use physical PTZ command.

In EPTZ mode, you may need to input stream index by set the high word of `dwCamera`, the low word is camera index, the stream index is start from 1.

**Requirements**

ServerManager.h

**See Also**

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#) [ServerManager\\_AbortProcess](#)

# ServerManager\_SetDOStatus

Set current DO information of the server. If DO is not to be set, put eSrvCtrlDiDoNone in the corresponding position of the array.

## Syntax

```
SCODE ServerManager_SetDOStatus ( HANDLE hDevice,  
                                  ESrvCtrlDiDoLevel * peDOStatus,  
                                  DWORD dwMaxDONum,  
                                  TSVRCTRL_NOTIFY * ptNotify,  
                                  HANDLE * phProc );
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### peDOStatus

[in] the pointer to a [ESrvCtrlDiDoLevel](#) to update the DO status of the server. It is the pointer of the first element of the DO status array.

### dwMaxDONum

[in]the size value of DO status array

### ptNotify

[in] the pointer [TSVRCTRL\\_NOTIFY](#) contains the callback function and related information. It cannot be null and the content should be valid.

### phProc

[out] the pointer to a http operation handle. The http operation handle is used to read more data. The module will pass a handle to the caller if there is more data to be read. If none, the phProc will be null. The users do not need to free the http operation handle. It is kept internally

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will return soon. And callback function will be raised when the operation is really done.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#) [ServerManager\\_AbortProcess](#)

VIVOTEK CONFIDENTIAL  
2009.08.31

# ServerManager\_SetMotionDetectionInfo

Set current motion detection setting information of the server.

## Syntax

```
SCODE  
ServerManager_SetMotionDetectionInfo (  
    HANDLE hDevice,  
    DWORD dwCamera,  
    const TSRVCTRL_MODETECT_INFO  
        *ptMDInfo  
    TSRVCTRL_NOTIFY * ptNotify,  
    HANDLE * phProc  
);
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### dwCamera

[in] in the multiple camera model, it indicate the order from the right with starting as 1. Otherwise the module will ignore it.

### ptMDInfo

[in] the pointer to a [TSRVCTRL\\_MODETECT\\_INFO](#) to store the motion detection to update to the server.

### ptNotify

[in] the pointer [TSRVCTRL\\_NOTIFY](#) contains the callback function and related information. It cannot be null and the content should be valid.

### phProc

[out] the pointer to a http operation handle. The http operation handle is used to read more data. The module will pass a handle to the caller if there is more data to be read. If none, the phProc will be null. The users do not need to free the http operation handle. It is kept internally

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will return soon. And callback function will be raised when the operation is really done.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#) [ServerManager\\_AbortProcess](#)

VIVOTEK CONFIDENTIAL  
2009.08.31

# ServerManager\_SetPrivateMaskInfo

Set current private mask setting information of the server.

## Syntax

```
SCODE  
ServerManager_SetPrivateMaskInfo ( HANDLE hDevice,  
                                   DWORD dwCamera,  
                                   const TSRVCTRL_PRIVATEMASK_INFO  
                                   * ptPMInfo  
                                   TSRVCTRL_NOTIFY * ptNotify,  
                                   HANDLE * phProc  
                                   );
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### dwCamera

[in] in the multiple camera model, it indicate the order from the right with starting as 1. Otherwise the module will ignore it.

### ptPMInfo

[out] the pointer to a [TSRVCTRL\\_PRIVATEMASK\\_INFO](#) to receive the private mask setting information to be update to the serve.

### ptNotify

[in] the pointer [TSRVCTRL\\_NOTIFY](#) contains the callback function and related information. It cannot be null and the content should be valid.

### phProc

[out] the pointer to a http operation handle. The http operation handle is used to read more data. The module will pass a handle to the caller if there is more data to be read. If none, the phProc will be null. The users do not need to free the http operation handle. It is kept internally

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will return soon. And callback function will be raised when the operation is really done.



## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#) [ServerManager\\_AbortProcess](#)

VIVOTEK CONFIDENTIAL  
2009.08.31

# ServerManager\_UartRead

Read responded data from device's serial port. Note that the server will not cache the UART data. So if the timing is incorrect, the data may lose.

## Syntax

```
SCODE ServerManager_UartRead ( HANDLE hDevice,
                                DWORD dwPort,
                                BYTE *pszResponse,
                                DWORD dwReadLen,
                                BOOL bFlush,
                                DWORD dwWaitTime,
                                TSVRCTRL_NOTIFY * ptNotify,
                                HANDLE * phProc );
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### dwPort

[in] the com port.

### pszResponse

[out] the pointer to the string to receive the response data

### dwReadLen

[in out] the pointer to a DWORD to represent the length of the buffer. After the function return, it will be replaced with the actual length value.

### bFlush

[in] the boolean value to the server should flush the Uart or not before it begin to read data..

### dwWaitTime

[in] the milliseconds unit value for the server to wait for the specified amount of data

### ptNotify

[in] the pointer [TSVRCTRL\\_NOTIFY](#) contains the callback function and related information. It cannot be null and the content should be valid.

### phProc

[out] the pointer to a http operation handle. The http operation handle is used to read more data. The module will pass a handle to the caller if there is more data to be read. If none, the phProc will be null. The users do not need to free the http operation handle. It is kept internally

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will return soon. And callback function will be raised when the operation is really done.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#) [ServerManager\\_AbortProcess](#)

## ServerManager\_UartReadAFWrite

Read responded data from device's serial port after send the command to the server. Note that the server will not cache the UART data. So if the timing is incorrect, the data may lose.

### Syntax

```
SCODE ServerManager_UartRead ( HANDLE hDevice,  
                                DWORD dwPort,  
                                const BYTE *pbyCommand,  
                                DWORD dwLen,  
                                BYTE *pszResponse,  
                                DWORD dwReadLen,  
                                BOOL bFlush,  
                                DWORD dwWaitTime,  
                                TSVRCTRL_NOTIFY * ptNotify,  
                                HANDLE * phProc );
```

### Parameters

#### **hDevice**

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

#### **dwPort**

[in] the com port.

#### **pbyCommand**

[in] the pointer to the command string.

#### **dwLen**

[in] the length value of the command.

#### **pszResponse**

[out] the pointer to the string to receive the response data

#### **dwReadLen**

[in out] the pointer to a **DWORD** to represent the length of the buffer. After the function return, it will be replaced with the actual length value.

#### **bFlush**

[in] the boolean value to the server should flush the Uart or not before it begin to read data..

#### **dwWaitTime**

[in] the milliseconds unit value for the server to wait for the specified amount of data

**ptNotify**

[in] the pointer [TSVRCTRL\\_NOTIFY](#) contains the callback function and related information. It cannot be null and the content should be valid.

**phProc**

[out] the pointer to a http operation handle. The http operation handle is used to read more data. The module will pass a handle to the caller if there is more data to be read. If none, the phProc will be null. The users do not need to free the http operation handle. It is kept internally

**Return Values**

Returns S\_OK if successful, or an error value otherwise.

**Remarks**

The function will return soon. And callback function will be raised when the operation is really done.

**Requirements**

ServerManager.h

**See Also**

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#) [ServerManager\\_AbortProcess](#)

# ServerManager\_UartWrite

Send data to device's serial port.

## Syntax

```
SCODE ServerManager_UartWrite (    HANDLE hDevice,  
                                   DWORD dwPort,  
                                   const BYTE *pbyCommand,  
                                   DWORD dwLen,  
                                   TSVRCTRL_NOTIFY * ptNotify,  
                                   HANDLE * phProc    );
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### dwPort

[in] the com port.

### pbyCommand

[in] the pointer to the command string.

### dwLen

[in] the length value of the command.

### ptNotify

[in] the pointer [TSVRCTRL\\_NOTIFY](#) contains the callback function and related information. It cannot be null and the content should be valid.

### phProc

[out] the pointer to a http operation handle. The http operation handle is used to read more data. The module will pass a handle to the caller if there is more data to be read. If none, the phProc will be null. The users do not need to free the http operation handle. It is kept internally

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will return soon. And callback function will be raised when the operation is really done.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#) [ServerManager\\_AbortProcess](#)

VIVOTEK CONFIDENTIAL  
2009.08.31

# ServerManager\_UpdateLocalConfig

Update the local system parameter. It means to retrieve the configuration file from server again. Any change made locally is overwritten.

## Syntax

```
SCODE ServerManager_UpdateLocalConfig ( HANDLE hDevice,  
                                         TSVRCTRL_NOTIFY * ptNotify );
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### ptNotify

[in] the pointer [TSVRCTRL\\_NOTIFY](#) contains the callback function and related information. It cannot be null and the content should be valid.

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will return soon. And callback function will be raised when the operation is really done.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#) [ServerManager\\_AbortProcess](#)



## ServerManager\_UpdateRemoteConfig

Update the remote system parameters. It means upload the local configuration file to remote server. The server will reboot after the file is uploaded by default. To prevent server rebooting, please set the eSystemResetSystem item to "No". But some change of the configuration will take effect only if server reboot, such as the new IP address.

### Syntax

```
SCODE ServerManager_UpdateRemoteConfig ( HANDLE hDevice,  
                                           TSVRCTRL_NOTIFY * ptNotify );
```

### Parameters

#### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

#### ptNotify

[in] the pointer [TSVRCTRL\\_NOTIFY](#) contains the callback function and related information. It cannot be null and the content should be valid.

### Return Values

Returns S\_OK if successful, or an error value otherwise.

### Remarks

The function will return soon. And callback function will be raised when the operation is really done.

### Requirements

ServerManager.h

### See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#) [ServerManager\\_AbortProcess](#)

## ServerManager\_AbortProcess

Abort the process of the non-block mode operation.

### Syntax

```
SCODE ServerManager_AbortProcess ( HANDLE hProc );
```

### Parameters

#### hProc

[in] the handle to the non-block mode operation handle. The handle is returned when calling non-blocking operation.

### Return Values

Returns S\_OK if successful, or an error value otherwise.

### Remarks

In ServerUtl applied case, this function will return ERR\_NOT\_IMPLEMENT.

### Requirements

ServerManager.h

## 4.6. Block mode API Definition

The blocking mode API definition is depicted here.

VIVOTEK CONFIDENTIAL  
2009.08.31

# ServerManager\_GetDIStatusBlock

Get current DI information of the server. This is a synchronous function and it will operate in block way

## Syntax

```
SCODE ServerManager_GetDIStatusBlock ( HANDLE hDevice,  
                                       ESrvCtrlDiDoLevel * peDIStatus,  
                                       DWORD *pdwMaxDINum );
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#)

### peDIStatus

[out] the pointer to a [ESrvCtrlDiDoLevel](#) array buffer to receive the retrieved DI status from the server.

### pdwMaxDINum

[in out] the pointer to a DWORD to represent the size value of DI status array buffer. And the module will replace it with the actual size value of DI status array. If the actual size is more than \*pdwMaxDINum, the module will return error. And the give buffer will not be filled

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will not return the result until the whole operation has been done. But you can use ServerManager\_StopRequest to abort it.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)

# ServerManager\_GetDOStatusBlock

Get current DO information of the server. This is a synchronous function and it will operate in block way

## Syntax

```
SCODE ServerManager_GetDOStatusBlock ( HANDLE hDevice,  
                                       ESrvCtrlDiDoLevel *  
                                       peDOStatus,  
                                       DWORD *pdwMaxDONum );
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### peDOStatus

[out] the pointer to a [ESrvCtrlDiDoLevel](#) array buffer to receive the retrieved DO status from the server.

### pdwMaxDONum

[in out] the pointer to a DWORD to represent the size value of DO status array buffer. And the module will replace it with the actual size value of DO status array. If the actual size is more than \*pdwMaxDONum, the module will return error. And the give buffer will not be filled

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will not return the result until the whole operation has been done. But you can use ServerManager\_StopRequest to abort it.

In ServerUtl applied case, this function may not find the appropriate one to do it. The module will return ERR\_NOT\_IMPLEMENT.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)

# ServerManager\_GetMotionDetectionInfoBlock

Get current motion detection setting information of the server. This is a synchronous function and it will operate in block way

## Syntax

<b>SCORE</b> <b>ServerManager_GetMotionDetectionInfoBlock (</b>	<b>HANDLE</b> hDevice, <b>DWORD</b> dwCamera, <b>TSRVCTRL_MODETECT_INFO *</b> ptMDInfo <b>);</b>
--	--

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### dwCamera

[in] in the multiple camera model, it indicate the order from the right with starting as 1. Otherwise the module will ignore it.

### ptMDInfo

[out] the pointer to a TSRVCTRL\_MODETECT\_INFO to receive the motion detection setting information .

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will not return the result until the whole operation has been done. But you can use ServerManager\_StopRequest to abort it.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)

# ServerManager\_GetPrivateMaskInfoBlock

Get current private mask setting information of the server. This is a synchronous function and it will operate in block way

## Syntax

<b>SCORE</b> <b>ServerManager_GetPrivateMaskInfoBlock (</b>	<b>HANDLE</b> hDevice, <b>DWORD</b> dwCamera, <b>TSRVCTRL_PRIVATEMASK_INFO *</b> ptPMInfo <b>);</b>
--	---

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### dwCamera

[in] in the multiple camera model, it indicate the order from the right with starting as 1. Otherwise the module will ignore it.

### ptPMInfo

[out] the pointer to a [TSRVCTRL\\_PRIVATEMASK\\_INFO](#) to receive the private mask setting information .

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will not return the result until the whole operation has been done. But you can use ServerManager\_StopRequest to abort it.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)

# ServerManager\_MoveCameraBlock

Move the camera by the specified coordinate. This is a synchronous function and it will operate in block way

## Syntax

```
SCODE ServerManager_MoveCameraBlock ( HANDLE hDevice,  
                                       DWORD dwCamera,  
                                       DWORD dwStream,  
                                       DWORD dwX,  
                                       DWORD dwY,  
                                       DWORD dwDisplayWidth,  
                                       DWORD dwDisplayHeight,  
                                       BOOL bWaitRes );
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### dwCamera

[in] in the multiple camera model, it indicates the order from the right with starting as 1. Otherwise it will be ignored.

### dwStream

[in] in the dual-stream camera model, it indicates resolution of which stream will be get for computing the target position.

### dwX

[in] the x coordinate for the new position. This position is related to the left-upper corner of current camera view. The camera will treat this value as the new center x coordinate. This value should be the normal size of camera video size

### dwY

[in] the y coordinate for the new position. This position is related to the left-upper corner of current camera view. The camera will treat this value as the new center y coordinate. This value should be the normal size of camera video size.

### dwDisplayWidth

[in] The weight of the display video size.

### dwDisplayHeight

[in] The height of the display video size.

### bWaitRes

[in] the boolean value to wait for the server's response or not.



## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

If you want to control the device or get some information from the device, you must open the device first.

The function will not return the result until the whole operation has been done. But you can use ServerManager\_StopRequest to abort it.

For supporting digital move command, you must set the device to EPTZ mode by call the ServerManager\_SetPTZType.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)

## ServerManager\_OpenDeviceBlock

Open the specified device to access some information. It will connect to the server and retrieve the server model and configuration file. This is a synchronous function and it will operate in block way

### Syntax

```
SCODE ServerManager_OpenDeviceBlcok ( HANDLE hDevice,  
                                     TCHAR *lpszModelName,  
                                     DWORD dwMaxSize );
```

### Parameters

**hDevice**

[in] the handle of the specified device. It returned from [ServerManager\\_CreateDevice](#).

**lpszModelName**

[in] the pointer the TCHAR to receive the retrieved model name

**dwMaxSize**

[in] the model name of the specified device

### Return Values

Returns S\_OK if successful, or an error value otherwise.

### Remarks

If you want to control the device or get some information from the device, you must open the device first.

The function will not return the result until the whole operation has been done. But you can use `ServerManager_StopRequest` to abort it.

### Requirements

ServerManager.h

### See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)

## ServerManager\_OpenDeviceBlockW

Open the specified device to access some information. It will connect to the server and retrieve the server model and configuration file. This is a synchronous function and it will operate in block way. This is wide-character version of ServerManager\_OpenDeviceBlock.

### Syntax

```
SCODE ServerManager_OpenDeviceBlcokW ( HANDLE hDevice,  
                                        wchar_t *lpszModelName,  
                                        DWORD dwMaxSize );
```

### Parameters

**hDevice**

[in] the handle of the specified device. It returned from [ServerManager\\_CreateDevice](#).

**lpszModelName**

[in] the pointer the wchar\_t to receive the retrieved model name

**dwMaxSize**

[in] the model name of the specified device

### Return Values

Returns S\_OK if successful, or an error value otherwise.

### Remarks

If you want to control the device or get some information from the device, you must open the device first.

The function will not return the result until the whole operation has been done. But you can use ServerManager\_StopRequest to abort it.

### Requirements

ServerManager.h

### See Also

[ServerManager\\_OpenDeviceBlock](#)

## ServerManager\_OpenDeviceLiteBlock

Open the specified device to access some information. It will connect to the server and retrieve the server model, but it will not to retrieve configuration file. You can retrieve configuration file later by using [ServerManager\\_UpdateLocalConfig](#) or [ServerManager\\_UpdateLocalConfigBlock](#). This is a synchronous function and it will operate in block way.

### Syntax

```
SCODE ServerManager_OpenDeviceLiteBlcok ( HANDLE hDevice,  
                                           TCHAR *lpszModelName,  
                                           DWORD dwMaxSize );
```

### Parameters

#### **hDevice**

[in] the handle of the specified device. It returned from [ServerManager\\_CreateDevice](#).

#### **lpszModelName**

[in] the pointer the TCHAR to receive the retrieved model name

#### **dwMaxSize**

[in] the model name of the specified device

### Return Values

Returns S\_OK if successful, or an error value otherwise.

### Remarks

If you want to control the device or get some information from the device, you must open the device first.

The function will not return the result until the whole operation has been done. But you can use [ServerManager\\_StopRequest](#) to abort it.

The function is faster than [ServerManager\\_OpenDeviceBlock](#), because it will not retrieve configuration file. But it will make some functions about processing configuration file failed until you call [ServerManager\\_UpdateLocalConfig](#) or [ServerManager\\_UpdateLocalConfigBlock](#).

### Requirements

ServerManager.h

### See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager](#)

VIVOTEK CONFIDENTIAL  
2009.08.31

## ServerManager\_OpenDeviceLiteBlockW

Open the specified device to access some information. It will connect to the server and retrieve the server model, but it will not to retrieve configuration file. You can retrieve configuration file later by using `ServerManager_UpdateLocalConfig` or `ServerManager_UpdateLocalConfigBlock`. This is a synchronous function and it will operate in block way. This is wide-character version of `ServerManager_OpenDeviceLiteBlock`.

### Syntax

```
SCODE ServerManager_OpenDeviceLiteBlcokW ( HANDLE hDevice,  
                                           wchar_t *lpszModelName,  
                                           DWORD dwMaxSize );
```

### Parameters

#### **hDevice**

[in] the handle of the specified device. It returned from [ServerManager\\_CreateDevice](#).

#### **lpszModelName**

[in] the pointer the `wchar_t` to receive the retrieved model name

#### **dwMaxSize**

[in] the model name of the specified device

### Return Values

Returns `S_OK` if successful, or an error value otherwise.

### Remarks

If you want to control the device or get some information from the device, you must open the device first.

The function will not return the result until the whole operation has been done. But you can use `ServerManager_StopRequest` to abort it.

The function is faster than `ServerManager_OpenDeviceBlock`, because it will not retrieve configuration file. But it will make some functions about processing configuration file failed until you call `ServerManager_UpdateLocalConfig` or `ServerManager_UpdateLocalConfigBlock`.

### Requirements

`ServerManager.h`

## See Also

[ServerManager\\_OpenDeviceLiteBlock](#)

VIVOTEK CONFIDENTIAL  
2009.08.31

## ServerManager\_ReadDataBlock

This function is used to read the data from a http connection. User must ensure that the connection is already connected. Or error will be returned. If the data is still not enough, the same success code is returned. Use this function again to read further data.

The user must make sure the http connection is already established. Or it will return error.

### Syntax

<b>SCODE</b> ServerManager_ReadDataBlock ( <b>HANDLE</b> hOper, <b>BYTE</b> *pbyBuffer, <b>DWORD</b> *pdwBufLen                    );
---

### Parameters

#### hOper

[in] the http operation handle. It returned from [ServerController\\_SendHttpCommand](#) or [ServerController\\_SendHttpCommandReadBlock](#)

#### pbyBuffer

[out] the pointer to buffer to receive the http response data

#### pdwBufLen

[in out] the pointer to a DWORD to represent the length of the buffer. After the function return, it will be replaced with the actual length value

### Return Values

Returns S\_OK if successful, or an error value otherwise.

### Remarks

In ServerUtil applied case, this function may not find the appropriate one to do it. The module will return ERR\_NOT\_IMPLEMENT.

### Requirements

ServerManager.h

### See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)



## ServerManager\_SendHttpCommandBlock

This function is used to send an http command to server. For POST, the parameter is also put after the question mark as GET does. The result of the command is ignored by this function.

If need to read response data, please use `ServerController_SendHttpCommandReadBlock`.

This is a synchronous function and it will operate in block way

### Syntax

<b>SCORE</b> <b>ServerManager_SendHttpCommandBlock (</b>	<b>HANDLE</b> hDevice, <b>const TCHAR</b> *pszCommand, <b>BOOL</b> bPost <b>);</b>
---	---

### Parameters

#### **hDevice**

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

#### **pszCommand**

[in] the string pointer to an http command.

#### **bPost**

[in] Is the command sends by POST or GET? True is for POST, false for GET.

### Return Values

Returns `S_OK` if successful, or an error value otherwise.

### Remarks

The function will not return the result until the whole operation has been done. But you can use `ServerManager_StopRequest` to abort it.

### Requirements

ServerManager.h

### See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)

## ServerManager\_SendHttpCommandBlockW

This function is used to send an http command to server. For POST, the parameter is also put after the question mark as GET does. The result of the command is ignored by this function.

If need to read response data, please use `ServerController_SendHttpCommandReadBlock` or `ServerController_SendHttpCommandReadBlockW` for wide-character version.

This is a synchronous function and it will operate in block way. This is wide-character version of `ServerManager_SendHttpCommandBlock`.

### Syntax

<b>SCORE</b> <b>ServerManager_SendHttpCommandBolckW (</b>	<b>HANDLE</b> hDevice, <b>const wchar_t</b> *pszCommand, <b>BOOL</b> bPost <b>);</b>
--	---

### Parameters

#### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

#### pszCommand

[in] the string pointer to an http command.

#### bPost

[in] Is the command sends by POST or GET? True is for POST, false for GET.

### Return Values

Returns `S_OK` if successful, or an error value otherwise.

### Remarks

The function will not return the result until the whole operation has been done. But you can use `ServerManager_StopRequest` to abort it.

### Requirements

ServerManager.h

### See Also

[ServerManager\\_SendHttpCommandBlock](#)

# ServerManager\_SendHttpCommandReadBlock

This function is used to send an http command to server. For POST, the parameter is also put after the question mark as GET does. The result is read into pbyData until reaches the buffer length or connection is closed. If there are more data, special success code is returned and the phOper could hold the operation handle. Use ServerManager\_ReadDataBlock to read futher data.

This is a synchronous function and it will operate in block way

## Syntax

<b>SCORE</b> <b>ServerManager_SendHttpCommandReadBlock (</b>	<b>HANDLE</b> hDevice, <b>const TCHAR</b> *pszCommand, <b>BOOL</b> bPost <b>BYTE</b> *pbyData, <b>DWORD</b> *pdwBuflen, <b>HANDLE</b> *phOper <b>);</b>
---	---

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### pszCommand

[in] the string pointer to an http command.

### bPost

[in] Is the command sends by POST or GET? True is for POST, false for GET.

### pbyData

[out] the pointer to buffer to receive the http response data

### pdwBuflen

[in out] the pointer to a DWORD to represent the length value of buffer. It will be replaced with the length value of the response data.

### phOper

[out] the pointer to a http operation handle. The http operation handle is used to read more data. The module will pass a handle to the caller if there is more data to be read. If none, the phOper will be null. The users do not need to free the http operation handle. It is kept internally.

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will not return the result until the whole operation has been done. But you can use `ServerManager_StopRequest` to abort it.

## Requirements

`ServerManager.h`

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)

VIVOTEK CONFIDENTIAL  
2009.08.31

## ServerManager\_SendHttpCommandReadBlockW

This function is used to send an http command to server. For POST, the parameter is also put after the question mark as GET does. The result is read into pbyData until reaches the buffer length or connection is closed. If there are more data, special success code is returned and the phOper could hold the operation handle. Use ServerManager\_ReadDataBlock or ServerManager\_ReadDataBlockW for wide-character version to read further data.

This is a synchronous function and it will operate in block way. This is wide-character version of ServerManager\_SendHttpCommandReadBlock.

### Syntax

<b>SCODE</b> <b>ServerManager_SendHttpCommandReadBlockW (</b>	<b>HANDLE</b> hDevice, <b>const wchar_t</b> *pszCommand, <b>BOOL</b> bPost <b>BYTE</b> *pbyData, <b>DWORD</b> *pdwBuflen, <b>HANDLE</b> *phOper <b>);</b>
--	---

### Parameters

#### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

#### pszCommand

[in] the string pointer to an http command.

#### bPost

[in] Is the command sends by POST or GET? True is for POST, false for GET.

#### pbyData

[out] the pointer to buffer to receive the http response data

#### pdwBuflen

[in out] the pointer to a DWORD to represent the length value of buffer. It will be replaced with the length value of the response data.

#### phOper

[out] the pointer to a http operation handle. The http operation handle is used to read more data. The module will pass a handle to the caller if there is more data to be read. If none, the phOper will be null. The users do not need to free the http operation handle. It is kept internally.

### Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will not return the result until the whole operation has been done. But you can use `ServerManager_StopRequest` to abort it.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_SendHttpCommandReadBlock](#)

VIVOTEK CONFIDENTIAL  
2009.08.31

# ServerManager\_SendPTZCommandBlock

Send the PTZ command to the server. This is a synchronous function and it will operate in block way

## Syntax

<b>SCODE</b> <b>ServerManager_SendPTZCommandBlock(</b>	<b>HANDLE</b> hDevice, <b>DWORD</b> dwCamera, <b>ESRVCTRL_PTZ_COMMAND</b> PtzCommand, <b>const TCHAR *</b> pszExtraCmd, <b>BOOL</b> bWaitRes <b>);</b>
---	--

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### dwCamera

[in] in the multiple camera model, it indicates the order from the right with starting as 1. Otherwise it will be ignored.

### ePtzCommand

[in] the [ESRVCTRL\\_PTZ\\_COMMAND](#) to represent a command will be sent to the target device. Not all commands are accepted in every camera, so you may check the user's manual to make sure it.

### pszExtraCmd

[in] the extra information for certain PTZ command. Here is a list for the command

- eSrvCtrlPTZPresetAdd: the pointer to the name of the new preset location
- eSrvCtrlPTZPresetDel: the pointer to the name of preset location to be deleted
- eSrvCtrlPTZPresetRecall: the pointer to the name of the preset location to be recalled
- eSrvCtrlPTZPanSpeed: the pointer to the pan speed in string (number presented in string format)
- eSrvCtrlPTZTiltSpeed: the pointer to the tilt speed in string (number presented in string format)
- eSrvCtrlPTZFocusSpeed: the pointer to the tilt speed in string (number presented in string format)
- eSrvCtrlPTZCustom: the pointer to the customized command ID in string

(number presented in string format)

**bWaitRes**

[in] the boolean value to wait for the server's response or not.

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will not return the result until the whole operation has been done. But you can use `ServerManager_StopRequest` to abort it.

For supporting digital PTZ command, you must set the device to EPTZ mode by call the `ServerManager_SetPTZType`, otherwise the device will use physical PTZ command.

In EPTZ mode, you may need to input stream index by set the high word of `dwCamera`, the low word is camera index, the stream index is start from 1.

## Requirements

`ServerManager.h`

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)



# ServerManager\_SetDOStatusBlock

Set current DO information of the server. This is a synchronous function and it will operate in block way

## Syntax

<b>SCODE</b> <b>ServerManager_SetDOStatusBlock (</b>	<b>HANDLE</b> hDevice, <b>ESrvCtrlDiDoLevel</b> * peDOStatus, <b>DWORD</b> dwMaxDONum, <b>BOOL</b> bWaitRes <b>);</b>
---	---

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### peDOStatus

[in] the pointer to a [ESrvCtrlDiDoLevel](#) to update the DO status of the server. It is the pointer of the first element of the DO status array.

### dwMaxDONum

[in]the size value of DO status array

### bWaitRes

[in] the Boolean value to wait for the server's response or not.

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will not return the result until the whole operation has been done. But you can use [ServerManager\\_StopRequest](#) to abort it.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)

# ServerManager\_SetMotionDetectionInfoBlock

Set current motion detection setting information of the server. This is a synchronous function and it will operate in block way

## Syntax

<b>SCODE</b> <b>ServerManager_SetMotionDetectionInfoBlock (</b>	<b>HANDLE</b> hDevice, <b>DWORD</b> dwCamera, <b>TSRVCTRL_MODETECT_INFO *</b> ptMDInfo, <b>BOOL</b> bWaitRes <b>);</b>
--	---

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### dwCamera

[in] in the multiple camera model, it indicate the order from the right with starting as 1. Otherwise the module will ignore it.

### ptMDInfo

[in] the pointer to a [TSRVCTRL\\_MODETECT\\_INFO](#) to store the motion detection to update to the server.

### bWaitRes

[in] the Boolean value to wait for the server's response or not.

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will not return the result until the whole operation has been done. But you can use [ServerManager\\_StopRequest](#) to abort it.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)

# ServerManager\_SetPrivateMaskInfoBlock

Set current private mask setting information of the server. This is a synchronous function and it will operate in block way

## Syntax

<b>SCODE</b> <b>ServerManager_SetPrivateMaskInfoBlock (</b>	<b>HANDLE</b> hDevice, <b>DWORD</b> dwCamera, <b>TSRVCTRL_PRIVATEMASK_INFO</b> *ptPMInfo, <b>BOOL</b> bWaitRes <b>);</b>
--	---

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### dwCamera

[in] in the multiple camera model, it indicate the order from the right with starting as 1. Otherwise the module will ignore it.

### ptPMInfo

[out] the pointer to a [TSRVCTRL\\_PRIVATEMASK\\_INFO](#) to receive the private mask setting information.

### bWaitRes

[in] the Boolean value to wait for the server's response or not.

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will not return the result until the whole operation has been done. But you can use [ServerManager\\_StopRequest](#) to abort it.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)

## ServerManager\_UartReadBlock

Read responded data from device's serial port. Note that the server will not cache the UART data. So if the timing is incorrect, the data may lose. This is a synchronous function and it will operate in block way

### Syntax

```
SCODE ServerManager_UartReadBlock ( HANDLE hDevice,  
                                     DWORD dwPort,  
                                     BYTE *pszResponse,  
                                     DWORD *pdwReadLen,  
                                     BOOL bFlush,  
                                     DWORD dwWaitTime, );
```

### Parameters

#### **hDevice**

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

#### **dwPort**

[in] the com port.

#### **pszResponse**

[out] the pointer to the string to receive the response data

#### **pdwReadLen**

[in out] the pointer to a **DWORD** to represent the length of the buffer. After the function return, it will be replaced with the actual length value.

#### **bFlush**

[in] the Boolean value to the server should flush the Uart or not before it begin to read data..

#### **dwWaitTime**

[in] the milliseconds unit value for the server to wait for the specified amount of data

### Return Values

Returns **S\_OK** if successful, or an error value otherwise.

### Remarks

The function will not return the result until the whole operation has been done. But you can use **ServerManager\_StopRequest** to abort it.

## Requirements

ServerManager.h

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)

VIVOTEK CONFIDENTIAL  
2009.08.31

## ServerManager\_UartReadAFWriteBlock

Read responded data from device's serial port after send the command to the server. Note that the server will not cache the UART data. So if the timing is incorrect, the data may lose. This is a synchronous function and it will operate in block way

### Syntax

```
SCODE ServerManager_UartReadAFWriteBlock ( HANDLE hDevice,
                                           DWORD dwPort,
                                           onst BYTE *pbyCommand,
                                           DWORD dwLen,
                                           BYTE *pszResponse,
                                           DWORD dwReadLen,
                                           BOOL bFlush,
                                           DWORD dwWaitTime, );
```

### Parameters

**hDevice**

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

**dwPort**

[in] the com port.

**pbyCommand**

[in] the pointer to the command string.

**dwLen**

[in ] the length value of the command.

**pszResponse**

[out] the pointer to the string to receive the response data

**dwReadLen**

[in out] the pointer to a DWORD to represent the length of the buffer. After the function return, it will be replaced with the actual length value.

**bFlush**

[in] the Boolean value to the server should flush the Uart or not before it begin to read data..

**dwWaitTime**

[in] the milliseconds unit value for the server to wait for the specified amount of data

### Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will not return the result until the whole operation has been done. But you can use `ServerManager_StopRequest` to abort it.

## Requirements

`ServerManager.h`

## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)

VIVOTEK CONFIDENTIAL  
2009.08.31

# ServerManager\_UartWriteBlock

Send data to device's serial port.. This is a synchronous function and it will operate in block way.

## Syntax

```
SCODE ServerManager_UartWriteBlock ( HANDLE hDevice,  
                                     DWORD dwPort,  
                                     const BYTE *pbyCommand,  
                                     DWORD dwLen,  
                                     TSVRCTRL_NOTIFY * ptNotify,  
                                     HANDLE * phProc,  
                                     BOOL bWaitRes );
```

## Parameters

### hDevice

[in] the handle of the targeted device. It returned from [ServerManager\\_CreateDevice](#).

### dwPort

[in] the com port.

### pbyCommand

[in] the pointer to the command string.

### dwLen

[in ] the length value of the command.

### bWaitRes

[in] the boolean value to wait for the server's response or not.

## Return Values

Returns S\_OK if successful, or an error value otherwise.

## Remarks

The function will not return the result until the whole operation has been done. But you can use ServerManager\_StopRequest to abort it.

## Requirements

ServerManager.h



## See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)

VIVOTEK CONFIDENTIAL  
2009.08.31

## ServerManager\_UpdateLocalConfigBlock

Update the local system parameter. It means to retrieve the configuration file from server again. Any change made locally is overwritten. This is a synchronous function and it will operate in block way

### Syntax

```
SCODE ServerManager_UpdateLocalConfigBlock( HANDLE hDevice );
```

### Parameters

#### hDevice

[in] the handle of the specified device. It returned from [ServerManager\\_CreateDevice](#).

### Return Values

Returns S\_OK if successful, or an error value otherwise.

### Remarks

The function will not return the result until the whole operation has been done. But you can use ServerManager\_StopRequest to abort it.

### Requirements

ServerManager.h

### See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)

## ServerManager\_UpdateRemoteConfigBlock

Update the remote system parameters. It means upload the local configuration file to remote server. The server will reboot after the file is uploaded by default. To prevent server rebooting, please set the eSystemResetSystem item to "No". But some change of the configuration will take effect only if server reboot, such as the new IP address.

This is a synchronous function and it will operate in block way

### Syntax

```
SCODE ServerManager_UpdateRemoteConfigBlock( HANDLE hDevice );
```

### Parameters

#### hDevice

[in] the handle of the specified device. It returned from [ServerManager\\_CreateDevice](#).

### Return Values

Returns S\_OK if successful, or an error value otherwise.

### Remarks

The function will not return the result until the whole operation has been done. But you can use ServerManager\_StopRequest to abort it.

### Requirements

ServerManager.h

### See Also

[ServerManager\\_CreateDevice](#) [ServerManager\\_SetDeviceProperty](#) [ServerManager\\_OpenDevice](#) [ServerManager\\_OpenDeviceBlock](#)

# 5. Append

## 5.1. Error Code

### **SVRMNGR\_E\_TOOMANY\_DEVICE - 0x80090000**

It has created too many devices in ServerManager

### **SVRMNGR\_E\_NOTLOAD\_SRVCTRLLIB - 0x80090001**

ServerController library is not loaded successfully

### **SVRMNGR\_E\_NOTLOAD\_SRVUTILLIB - 0x80090002**

ServerUtility library is not loaded successfully

### **SVRMNGR\_E\_DEVICE\_NOT\_OPEN - 0x80090003**

ServerManager is not open device yet (not call ServerManager\_OpenDeviceBlock or ServerManager\_OpenDevice)

### **SVRMNGR\_E\_PARTIAL\_NOT\_IMPLEMENT - 0x80090004**

It may not find the appropriate function to work in some conditions

### **SVRMNGR\_E\_NOT\_MAPPING - 0x80090005**

It can not find corresponding ServerUtility config item index of parameter item in RX (new model)

### **SVRMNGR\_E\_KEY\_NOT\_FOUND - 0x80090006**

It can not find the item in functions ServerManager\_SetValueByNameBlock and ServerManager\_GetValueByName

### **SVRMNGR\_W\_MUST\_UPDATEREMOTE\_FIRST - 0x80090007**

This is warning that you may do ServerManager\_UpdateRemoteConfigBlock or ServerManager\_UpdateRemoteConfig first

**SVRMNGR\_E\_TIMEOUT - 0x80090008**

ServerManager occur timeout

**SVRMNGR\_E\_AUTH - 0x80090009**

It occurs authentication failed when trying to connect to server

**SVRMNGR\_E\_CONNECT\_FAILED - 0x8009000a**

ServerManager connect to server failed.

**SVRMNGR\_E\_PAGE\_NOT\_FOUND - 0x8009000b**

It can not find the page in Http request

**SVRMNGR\_E\_ERROR\_MODEL - 0x8009000c**

Use wrong library between ServerController and ServerUtility.

**SVRMNGR\_E\_CONVERT\_UNITOMB - 0x8009000d**

It occurs error when converting Unicode to multi-byte string.

**SVRMNGR\_E\_READ\_DATA\_ERROR - 0x8009000e**

The data that read from server is something wrong

**SVRMNGR\_E\_FTP\_GETFILE - 0x8009000f**

Get data from ftp server failed.

**SVRMNGR\_E\_FTP\_PETFILE - 0x80090010**

Put data to ftp server failed.

**SVRMNGR\_E\_OPENFILE - 0x80090011**

Open file failed.

**SVRMNGR\_E\_ABORTING - 0x80090012**

The request is be aborted.

**SVRMNGR\_E\_LOAD\_LIBRARY - 0x80090013**

Both libraries are not loaded.

**SVRMNGR\_E\_TOOMANY\_NONBLOCK\_OPER - 0x80090014**

Too many non-blocking operations are proceeding.

VIVOTEK CONFIDENTIAL  
2009.08.31