

# ETHIRIS<sup>®</sup>

VIDEO MANAGEMENT SOFTWARE

Integration with Ethiris 

Camera 4

Camera 5

Camera



Camera



Camera 11

Camera

Copyright © 2020 Kentima AB

Reproduction of the content of this manual, whether in full or in part, is prohibited under the Swedish Act on Copyright in Literary and Artistic Works without the consent of the copyright holders. This prohibition applies to any form of reproduction by printing, copying, tape recording, transfer to electronic media, etc.

**Production and layout:** Kentima AB

**Version:** 11.2

**First edition:** March 2020

**Printing:** Kentima AB

**Trademarks:** **Ethisis** and **WideQuick** are registered trademarks. All other marks belong to their respective owners.

Kentima AB

Postal address:

PO Box 174

SE-245 22 STAFFANSTORP

Street address:

SE-Kastanjevägen 4

S-245 44 STAFFANSTORP

Email address:

[info@kentima.com](mailto:info@kentima.com)

Internet:

[www.kentima.com](http://www.kentima.com)

# Contents

<b>1 Introduction</b>	<b>1:1</b>
1.1 Introduction .....	1:1
1.1.1 Use .....	1:1
1.2 General Description .....	1:1
<b>2 ActiveX</b>	<b>2:1</b>
2.1 Ethis ActiveX control .....	2:1
2.1.1 Overview .....	2:1
2.1.2 Installation .....	2:2
2.1.3 Testing the sample application .....	2:3
2.1.4 ActiveX control interface .....	2:7
<b>3 Remote control of Ethis Client</b>	<b>3:1</b>
3.1 Ethis Client Remote control .....	3:1
3.1.1 Overview .....	3:1
3.1.2 Installation .....	3:1
3.1.3 Testing the sample application .....	3:2
3.1.4 Remote control interface .....	3:9
<b>4 Modbus OPC Server</b>	<b>4:1</b>
4.1 Modbus OPC Server .....	4:1
4.1.1 Overview .....	4:1
4.1.2 Installation .....	4:1
4.1.3 Configuration .....	4:1
<b>5 Listeners</b>	<b>5:1</b>
5.1 Listeners .....	5:1
5.1.1 Overview .....	5:1
5.1.2 Listeners general .....	5:2
5.2 Types of listeners .....	5:3
5.2.1 TCP-inbound .....	5:3
5.2.2 TCP-outbound .....	5:4
<b>6 Explanation of Terms</b>	<b>6:1</b>
<b>7 Index</b>	<b>7:1</b>



<b>1 Introduction</b>	<b>1:1</b>
1.1 Introduction .....	1:1
1.1.1 Use .....	1:1
1.2 General Description .....	1:1



# 1 Introduction

## 1.1 Introduction

This manual is designed to help users integrate their Ethisis system with other kinds of systems.

There are a total of six different manuals for Ethisis. Besides this one, there are also *Installing Ethisis*, *Admin - Configuration for Ethisis*, *Ethisis Client User's Guide*, *Getting started with Ethisis* and *Getting started with Ethisis Mobile*.

This part of the manual suite describes some examples of how to integrate Ethisis with other systems. For information about the installation process of an Ethisis system, please refer to the *Installing Ethisis* manual. For information on how to get started with your Ethisis system after installation, please refer to the *Getting Started with Ethisis* manual.

For in-depth information on the various functions in Ethisis, please see the *Admin - Configuration for Ethisis* manual and the *Ethisis Client User's Guide* manual.

### 1.1.1 Use

The primary purpose of Ethisis is camera surveillance, which is performed in two ways. One way is manually to monitor live video from different cameras. The other way involves recording video from connected cameras. Recording can take place continuously from one or more cameras or in the form of video sequences when a recording condition is met. The recorded video can be played back afterward using sophisticated timelines and a VCR-like interface.

## 1.2 General Description

Ethisis is a surveillance system that uses network cameras and analog cameras, together with video encoders from different suppliers.

The product is divided into several program parts, where *Ethisis Server* and *Ethisis Client* are the most important ones. The server part manages all cameras and stores video on the hard disk. The client part displays live video and recorded video sequences.

There is also a program called *Ethisis Admin*, which is used for configuring all Ethisis modules in the system. In *Ethisis Admin*, you configure the *Ethisis Servers* by, e.g., defining which cameras are connected to each *Ethisis Server*, when video shall be recorded, what frame rate and resolution to use, etc. You also define the view layout in the various *Ethisis Clients* in the system using *Ethisis Admin*.

In theory, an unlimited number of cameras could be connected to each Ethis server and be displayed in the desired number of Ethis clients simultaneously. In practice, however, bandwidth and screen resolution set limits for the appropriate number of cameras connected.

There are different license levels for Ethis which permit different numbers of cameras to be connected. To meet the need for a large number of cameras, Ethis is designed with a focus on scalability. Scalability means that it is possible to divide your system into several Ethis servers and thus distribute the load over several computers.

A summary of the different Ethis programs is below:



#### ***Ethis Server***

Ethis Server is the core of an Ethis system. It handles all communication with the cameras and recording of video on a hard drive. Ethis Server runs as a service under the operating system and is automatically started as soon as the computer is started. An Ethis system can be comprised of one or several Ethis Servers.



#### ***Ethis Client***

Ethis Client is used for viewing live video and recorded video. An Ethis Client can connect to one or several Ethis Servers for access to cameras. An Ethis system can contain one or several Ethis Clients.



#### ***Ethis Admin***

Ethis Admin is used for configuring the different parts in an Ethis system. Ethis Server and Ethis Client are configured with this common tool. From any computer in the system, Ethis Admin can be run and used for configuring all Ethis components on-site.



#### ***Ethis Mobile***

*Ethis Mobile* is an app for connecting to your Ethis systems via your cell. With Ethis Mobile you can watch live video from various cameras in your system. You can also see all the alarms.



#### ***Ethis Server OPC Server***

Ethis Server OPC Server is a separate Ethis component that is used for letting other systems gain access to the information in Ethis Server. Any other system with an OPC client can connect to one or several Ethis Servers and read/write to all signals in Ethis Server. E.g. starting recording of a camera or control a PTZ camera.



#### ***Ethis ActiveX***

Ethis ActiveX is a component used for viewing live video from a camera connected to an Ethis Server. This component can be used in any system that can handle standard ActiveX components.



#### ***Ethis Viewer***

Ethis Viewer is a separate program used for viewing exported video from an Ethis system.



***Ethisis Signature Validator***

*Ethisis Signature Validator* is a separate software that is used to validate the authenticity of exported video files or jpg images, utilizing an embedded digital signature.

<b>2 ActiveX</b>	<b>2:1</b>
2.1 Ethisis ActiveX control.....	2:1
2.1.1 Overview.....	2:1
2.1.2 Installation.....	2:2
2.1.3 Testing the sample application.....	2:3
2.1.4 ActiveX control interface .....	2:7

## 2 ActiveX

### 2.1 Ethis ActiveX control

#### 2.1.1 Overview

The purpose of the Ethis ActiveX control is for letting other systems display live video from cameras connected to an Ethis Server. Any system that can handle standard ActiveX controls can use the Ethis ActiveX control.

The other system can run locally on the same computer as Ethis Server, or it can run on a remote computer. The vital thing to keep in mind is that the Ethis ActiveX control must be installed on the same computer where the system that will use the ActiveX control runs.

To be able to connect to an Ethis Server via the ActiveX control, the Ethis Server has to have the *ActiveX license option*. This is automatically included in the *Premium* license level and can be purchased specifically for *Extended* and *Advanced* license level. It is not at all available for Ethis Servers running at the *Basic* license level.

To check whether you have the *ActiveX* option, start *Kentima License Handler*, select *Ethis – Server* in the product list, and click *Next*. In the *License Information* dialog, there is a row with all *Options* for this license.

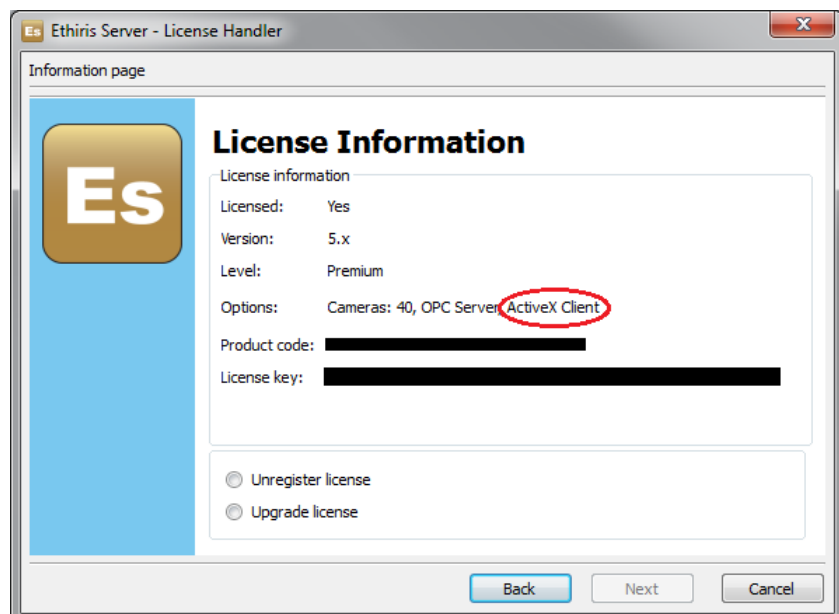


Figure 2.1 Make sure the license includes the ActiveX Client option

Click *Cancel* when you are done.

In the Ethisis setup, there is a specific *Setup Type* called *ActiveX Client* that can be selected when installing only the ActiveX control. If you want to install other Ethisis components together with the ActiveX control, you have to choose Setup Type *Custom* and then select the component *ActiveX Client* along with the other desired components.

There is also a *sample* in C++ .Net that shows how to use the ActiveX control.

## 2.1.2 Installation

Run Ethisis setup on the computer where you want the ActiveX control. In the following description of the sample, we assume you also install the *ActiveX client sample/C++ .Net* and that Visual Studio 2017 is installed on the computer if you want to look at the source files.

The ActiveX control will eventually connect to an Ethisis Server somewhere in the network. The Ethisis Server need not be installed on the same computer as the ActiveX control is, but it also works perfectly OK locally, with both the ActiveX control and Ethisis Server on the same computer.

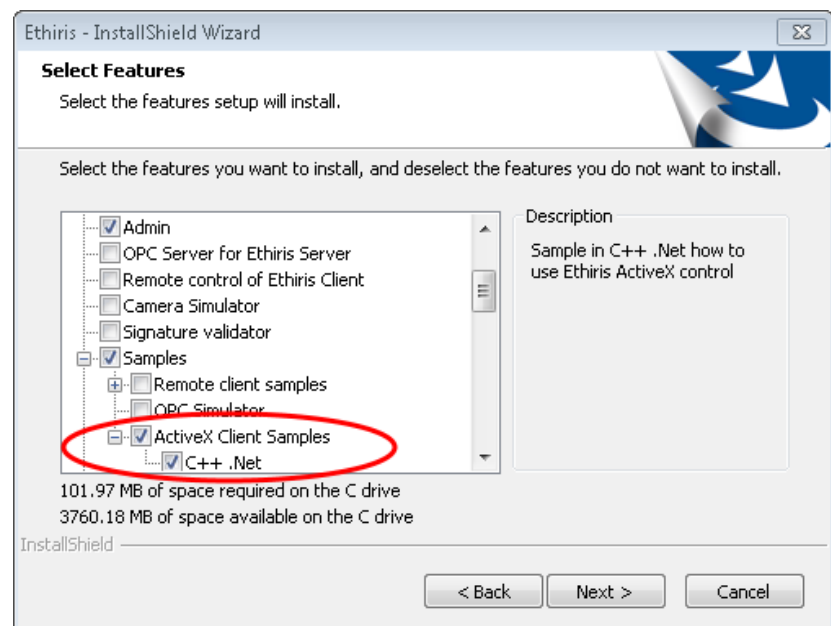


Figure 2.2 Select the components ActiveX Client Sample/C++ .Net & ActiveX Client.

The *ActiveX Client 32-bit (the actual ActiveX control)* is automatically selected when the sample is selected.



*The sample application is not intended to be used directly in a real application!*

### 2.1.3 Testing the sample application

After installation of the sample, you will find the sample files in a subfolder of the Ethisis installation folder. In general, this is `C:\Program Files (x86)\Kentima AB\Ethisis\Samples\ActiveXClient`.

Note that the sample application only is supposed to demonstrate how to use *ActiveX Client*. It is not intended to be used directly in a real application!

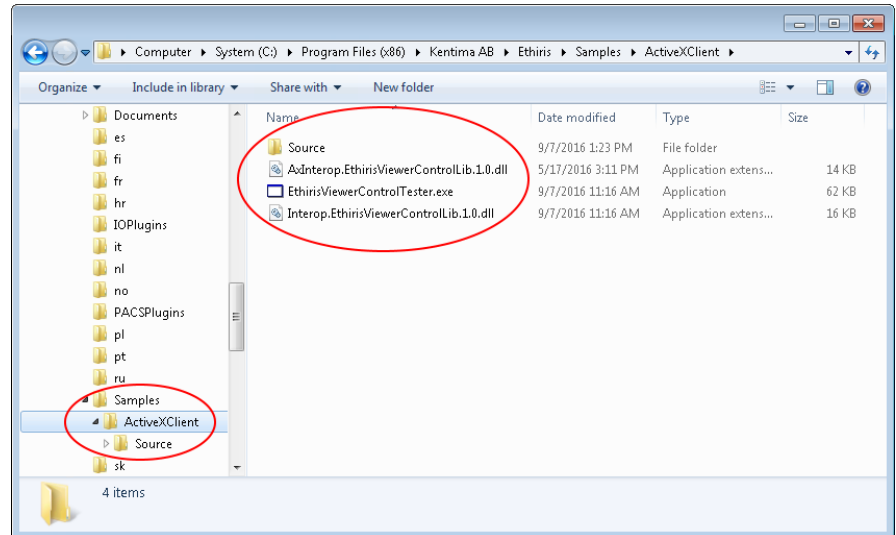


Figure 2.3 The ActiveX sample files in Windows Explorer.

The source files are in the folder *Source*. I assume that if you know Visual Studio 2015, you know how to use the source code files. Besides the source code files, there is a compiled version of the sample called *EthisisViewerControlTester.exe*.

Let's try it out!

**Double-click** the *EthisisViewerControlTester.exe* file in Windows Explorer. The application starts and looks like in *Figure 2.4*.

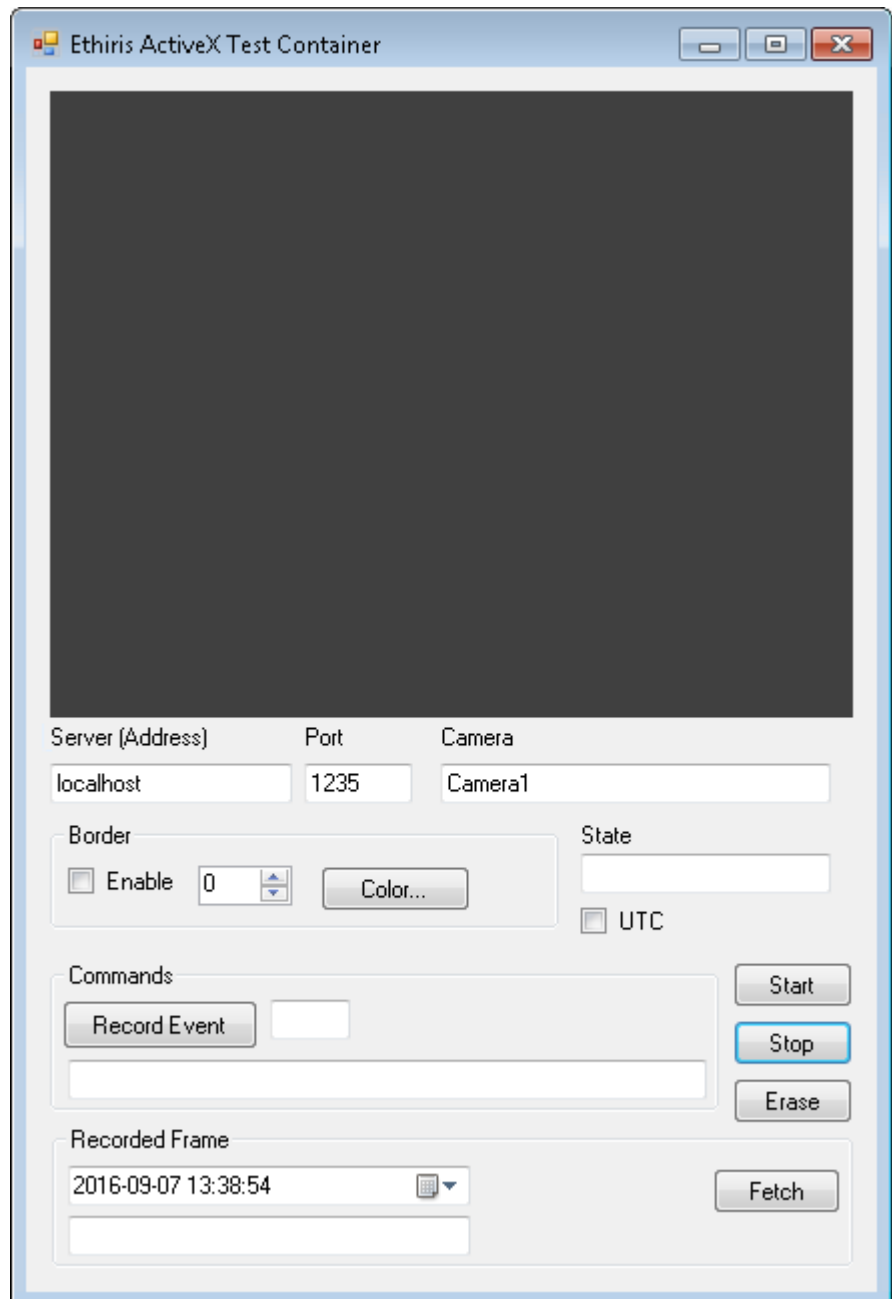


Figure 2.4 The ActiveX sample application just started.

At the top of the form is the ActiveX control, idle at the moment.

The idea is that we shall enter the name/IP address of an Ethisis Server and the name of a camera. Then we can start/stop a live video from the camera, and we can send commands to it.

If you have an Ethisis Server connected in Ethisis Admin, you can have a look there to get information about the Ethisis Server name/IP address and names of the connected cameras.

In *Figure 2.5*, the Ethisis Server panel is opened by double-clicking the Ethisis Server node in the tree view. In the panel to the right, the *computer name*, *IP-address*, and *port* are circled.

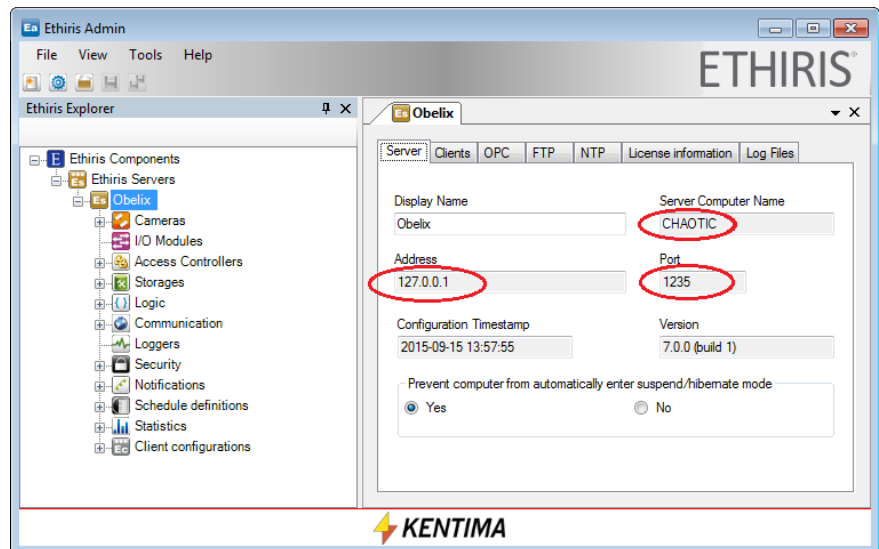


Figure 2.5 Ethisis Server configuration in Ethisis Admin.

Now, double-clicking the *Network Cameras* node in the tree view opens the corresponding panel and displays a list of connected cameras.

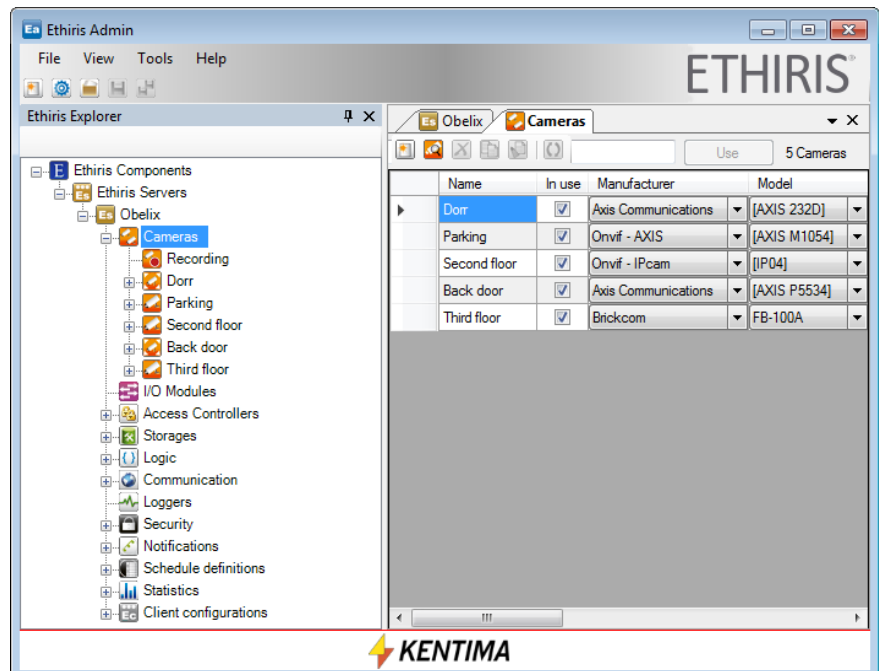


Figure 2.6 The Network Cameras list in Ethisis Admin.

Then we have enough information for trying the ActiveX sample.

**Enter** the appropriate *name/IP-address* of an Ethisis Server and a *camera name* in the section *Live video* in the sample application. In my case, I enter *chaotic* as Ethisis Server name and *Door* as camera name.

**Click** the *Start* button to start live video.

If the Ethisis Server runs on the same computer, you can leave *localhost* or enter *127.0.0.1* as Ethisis Server name since both these are aliases for the local computer.

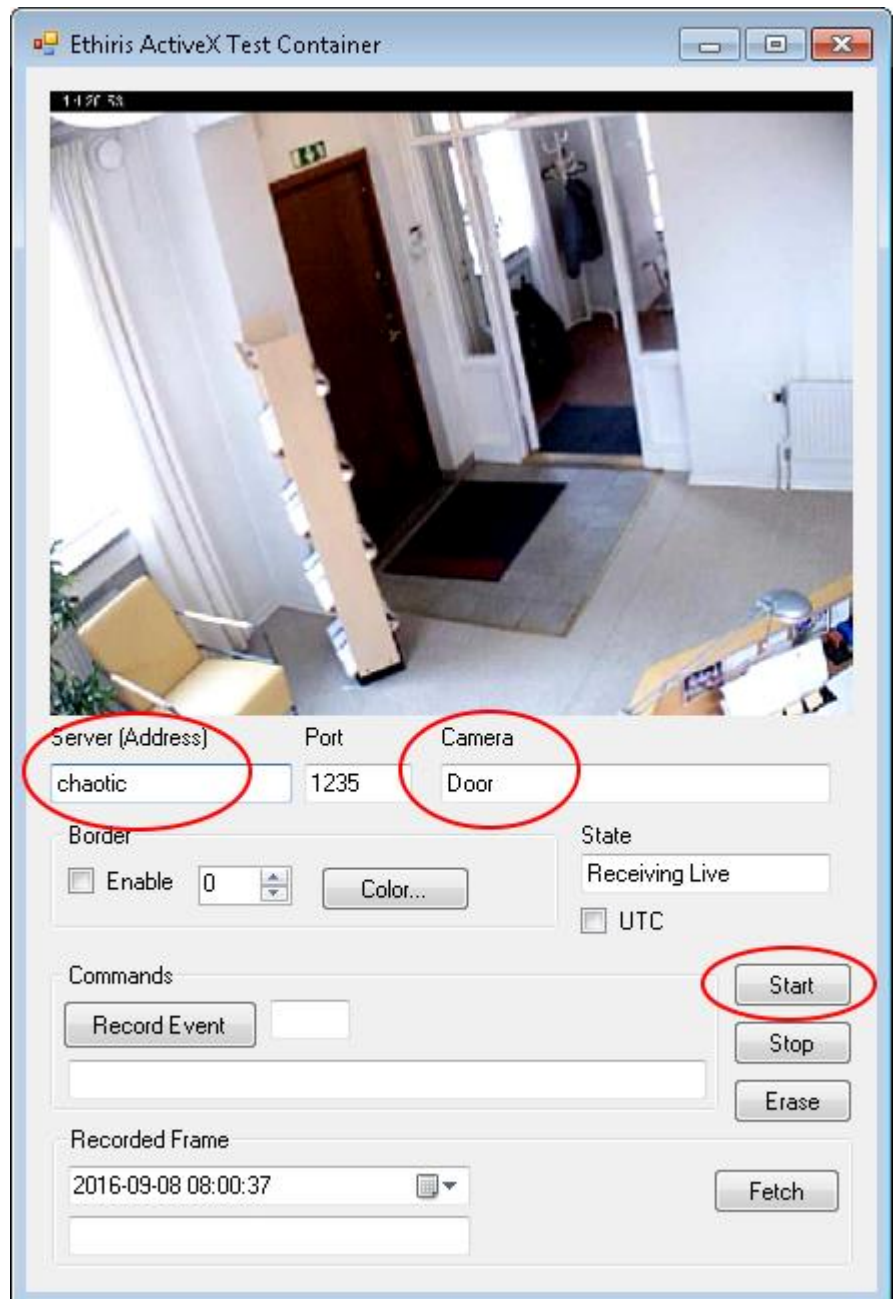


Figure 2.7 Live video in the test application.

Just click the *Stop* button for stopping the live video.

In the *Commands* section, you can start event recording for a camera. Click the *Record Event* button. The defined camera will start an event recording.

To the right of the *Record Event* button, is a field that displays the *Transaction ID* for the latest command. In the message field below, information is displayed about the latest command, such as *Transaction ID*, *Success status*, and *Command*.

For the moment, there is only one available command, *RecordEvent*. In the future, additional commands may be available, and then it makes more sense to have a transaction id identifying every command.



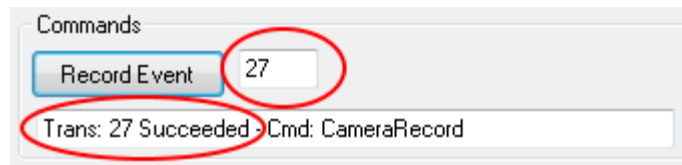


Figure 2.8 The Commands section from the test application.

In the section, *Border*, you can set a border for the ActiveX control and determine the *Border Color* and *Border Width*.

## 2.1.4 ActiveX control interface

When using the Ethisis ActiveX control, you use the interface of the control. The following is a reference to the various available properties, methods, events, and enumerations. These are the Ethisis specific properties, methods, and events. There are also a bunch of standard properties, methods, and events for an ActiveX control, like *Width* and *Height*.

### Properties

Name	Type	Description
<b>BorderColor</b>	OLE_COLOR	Determines the color of the border (if visible).
<b>BorderWidth</b>	Long	Determines the width of the border (if visible).
<b>BorderVisible</b>	Boolean	Determines if the border is visible or not.
<b>Camera</b>	String	Name of the preselected Camera in the Ethisis Server.
<b>ErrorDescription</b>	String	Error text (if in error state).
<b>LogItems</b>	Array	Reserved for future use
<b>Port</b>	Long	Preselected Port in Ethisis Server to connect to. This is almost always 1235.
<b>Server</b>	String	Address of the preselected Ethisis Server to connect to (IP address or DNS name).
<b>VideoState</b>	eVideoState	Current state of video communication with Ethisis Server. See <i>Enumerations</i> later in this document.
<b>UTC</b>	Boolean	Interprets timestamps as UTC time if <i>true</i> , otherwise as local time.

## Methods

### Erase() as Long

Clear the display area of the control.

### RecordEvent([Camera as String], [Server as String], [Port as Long]) as Long

Starts an event recording for the camera *Camera* on the Ethis Server *Server*. The parameters are optional, but if left out, the properties *Camera*, *Server*, and *Port* have to be set. The return value is the *TransactionID* for the command.

### RecordedFrame(Timestamp as String, [Margin as long], [Camera as String], [Server as String], [Port as Long]) as Long

Displays a recorded frame near the specified time, within the margin in ms (+/-) from the camera *Camera* on the Ethis Server *Server*. The parameters *Margin*, *Camera*, *Server*, and *Port* are optional, but if left out, the properties *Camera*, *Server*, and *Port* have to be set. The default value for *Margin* is 1000 ms. The return value is the *TransactionID* for the command. The parameter *Timestamp* uses the string format "YYYY-MM-DD hh:mm:ss[.fff]".

### StartLive([Camera as String], [Server as String], [Port as Long])

Starts to retrieve live video from the camera *Camera* on the Ethis Server *Server*. The parameters are optional, but if left out, the properties *Camera*, *Server*, and *Port* have to be set. The result is reported back via the event *VideoStateChanged*.

### Stop()

Stops retrieving live video from the current camera. The result is reported back via the event *VideoStateChanged*.

## Events

### CommandComplete(Success as Boolean, TransactionID as Long, Command as String, Message as String)

This event is sent from the ActiveX control when a command sent to it earlier is complete. The following parameters are sent in the event:

*Success* – true or false. If true, the command was completed successfully.

*TransactionID* – Contains the ID of the transaction for the command. This is the same ID that was returned from the *RecordEvent* method described above.

*Command* – The internal name of the command. For the moment, only one command is available, *CameraRecord*.

*Message* – If everything is ok, this parameter is empty. May contain a message from the ActiveX control about the command.

### LogValueBoolean (Name as String, Timestamp as String, Value as Boolean)

### LogValueDouble (Name as String, Timestamp as String, Value as Double)

**LogValueInteger** (Name as String, Timestamp as String, Value as Long)

**LogValueString** (Name as String, Timestamp as String, Value as String)

These events are reserved for future new functionality and are not used in the version.

**RecordedFrameDisplayed** (Timestamp as String, Width as long, Height as long)

This event is sent from the ActiveX control when a recorded video frame has been displayed. The following parameters are sent in the event:

*Timestamp* – Timestamp of the displayed video frame in the format “YYYY-MM-DD hh:mm:ss.fff”.

*Width* – Width in pixels of the displayed video frame.

*Height* – Height in pixels of the displayed video frame.

**VideoStateChanged**(NewState as eVideoState)

This event is sent from the ActiveX control when there is a change in the live video state, e.g., when the control starts receiving live video from the camera or when the live video stops. The following parameter is sent in the event:

*NewState* – There are a total of 7 different states as described below in the *Enumerations* section. The *NewState* parameter describes the new state of live video in the ActiveX control.

## Enumerations

**eVideoState** contains 7 different values from 0 to 6, as described below:

*vsIdle* – 0. No video is received, and the ActiveX control is idle.

*vsConnectingLive* – 1. Connect operation in progress for live video.

*vsConnectedLive* – 2. Connection established for live video.

*vsCreatingSessionLive* - 3. Call made to create a session. This has to do with the internal Ethisis protocol.

*vsRequestedLive* – 4. Sent request for live video from the specified camera.

*vsReceivingLive* – 5. Receiving live video. This is the normal state when live video is sent to the ActiveX control.

*vsError* – 6. Something is wrong.

*vsConnectingRecordedFrame* – 7. Connect operation in progress for the recorded video frame.

*vsConnectedRecordedFrame* – 8. Connection established for the recorded video frame.

*vsCreatingSessionRecordedFrame* – 9. Call made to create a session for the recorded video frame. This has to do with the internal Ethisis protocol.

*vsRequestedRecordedFrame* – 10. Sent request for recorded video frame from the specified camera.

*vsReceivingRecordedFrame* – 11. Receiving a recorded video frame.



<b>3 Remote control of Ethisis Client</b>	<b>3:1</b>
3.1 Ethisis Client Remote control.....	3:1
3.1.1 Overview.....	3:1
3.1.2 Installation.....	3:1
3.1.3 Testing the sample application.....	3:2
3.1.4 Remote control interface .....	3:9



# 3 Remote control of Ethisis Client

## 3.1 Ethisis Client Remote control

### 3.1.1 Overview

The purpose of the Ethisis Client Remote control is for allowing other systems to control one or several Ethisis Client regarding what is displayed as live and recorded video.

The other system can run locally on the same computer as Ethisis Client, or it can run on a remote computer. The vital thing to keep in mind is that the Ethisis Client Remote control must be installed on the same computer where the system that will use it runs.

In the Ethisis setup, there is a specific *Setup Type* called *Remote control of Ethisis Client* that can be selected when installing only the remote control. If you want to install other Ethisis components together with the remote control, you have to choose *Setup Type Custom* and then select the component *Remote control of Ethisis Client* along with the other desired components.

There are also two *samples* that show how to use the remote control; One in Visual Basic .Net 2017 and one in WideQuick.

### 3.1.2 Installation

Run Ethisis setup on the computer where you want the remote control. In the following description of the VB.Net sample, we assume you also install the *Remote client samples/VB.Net* and that Visual Basic .Net is installed on the computer.

The remote control will eventually connect to an Ethisis Client somewhere in the network. The Ethisis Client need not be installed on the same computer as the remote control is, but it also works perfectly OK locally, with both the remote control and Ethisis Client on the same computer.

If you want the sample project for *WideQuick*, just check the WideQuick checkbox in the *Select Features* dialog in the installation guide.

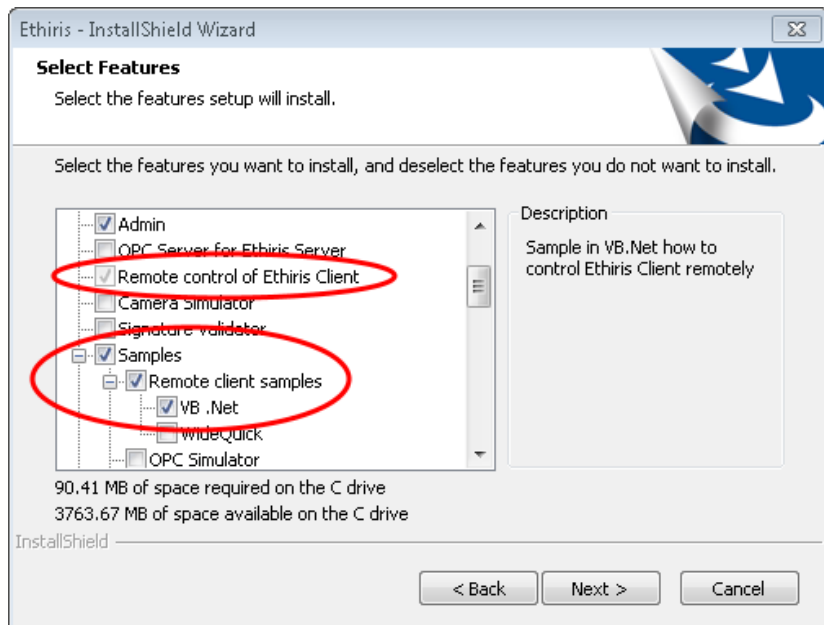


Figure 3.1 Select components Remote client sample & Remote control of Ethisis Client.

The *Remote control of Ethisis Client* component (the actual remote control) is automatically selected when the sample is selected.

### 3.1.3 Testing the sample application



*The sample application is not intended to be used directly in a real application!*

After installation of the sample, you find the sample files in a subfolder of the Ethisis installation folder. In general, this is `C:\Program Files\Kentima AB\Ethisis\Samples\RemoteClient\VB.Net`.

Note that the sample application only is supposed to demonstrate how to use *Remote control of Ethisis Client*. It is not intended to be used directly in a real application!

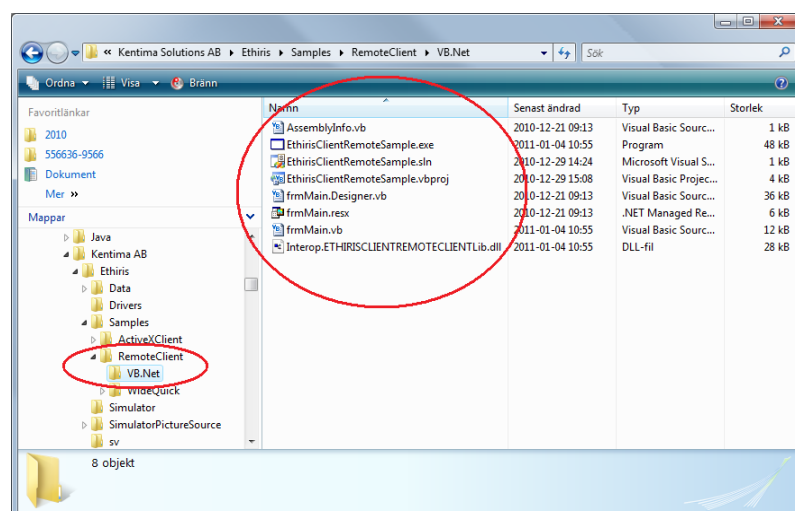


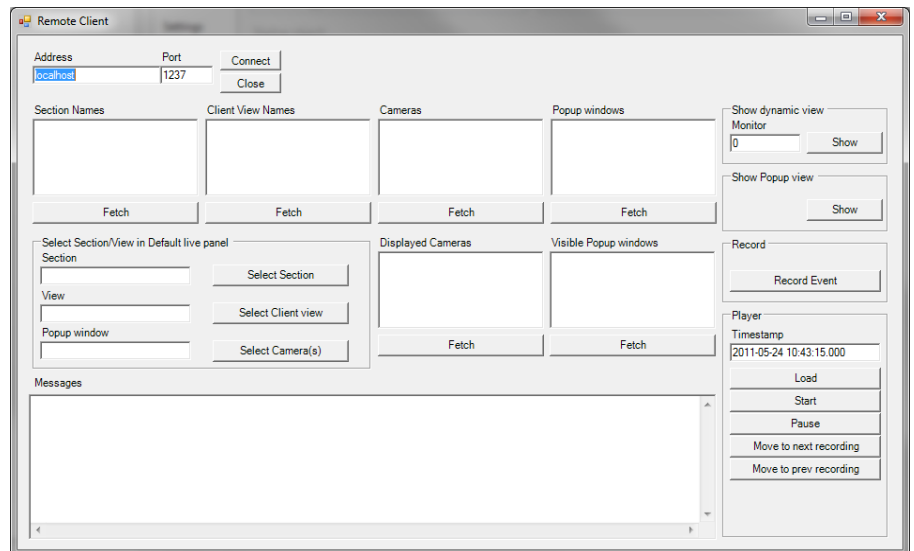
Figure 3.2 The Remote client sample files for VB.Net in Windows Explorer.

I assume that if you know Visual Basic .Net, you know how to use the source code files. Besides the source code files, there is a compiled version of the sample called *EthisisClientRemoteSample.exe*.

Let's try it out!



**Double-click** the *EthisisClientRemoteSample.exe* file in Windows Explorer. The application starts and looks like in *Figure 3.3*.

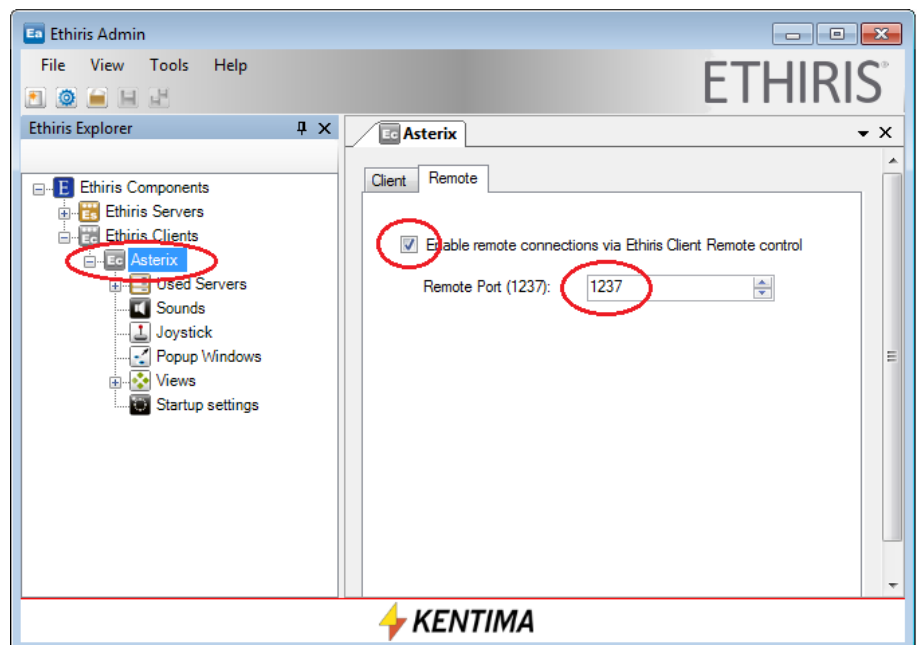


*Figure 3.3* The Remote client sample VB.Net application just started.

At the top of the form are fields for *Address* and *Port*. Default values are *localhost* and *1237*. *localhost* means the local computer. If the Ethisis Client you want to remote control runs on another computer, you have to enter the DNS name/IP-address of that computer instead.

Before you can connect to an Ethisis Client, you have to configure the client so that it listens for incoming connections on port 1237.

In Ethisis Admin, it looks like this:



*Figure 3.4* Enable remote connections on port 1237 in the client configuration.

Save the client configuration and then **start** *Ethisis Client*.

In my case, the Ethisis Client looks like below:

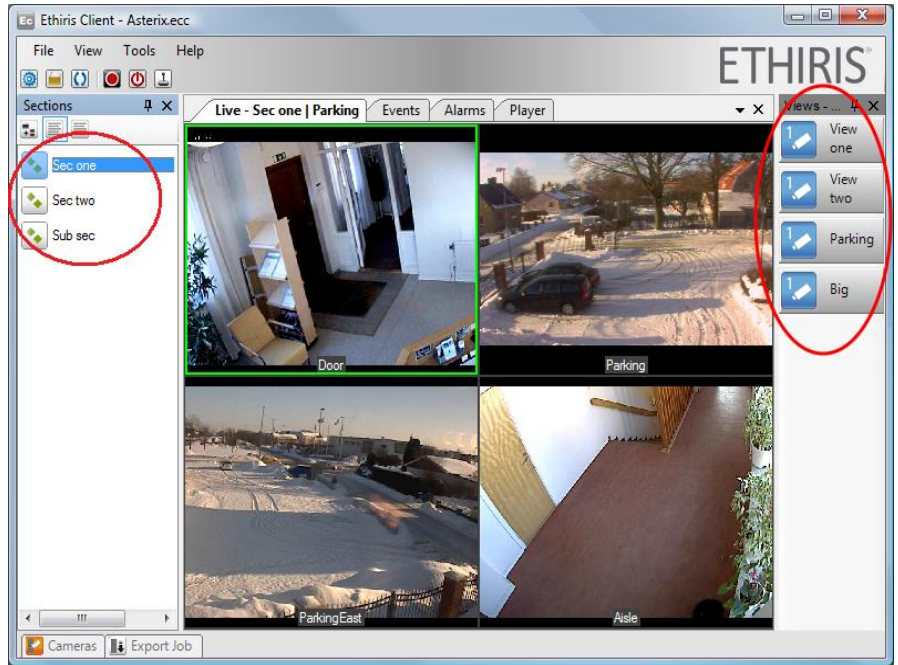


Figure 3.5 Ethisis Client with 3 different sections and 4 different views in the first section.

In the example above, there are three sections, called *Sec one*, *Sec two*, and *Sub sec*, and there are four views in the first section called *View one*, *View two*, *Parking*, and *Big*.

Now, it's time to connect to Ethisis Client from the sample application.

**Click** the *Connect* button. There should be a message informing you that the connection was successful.

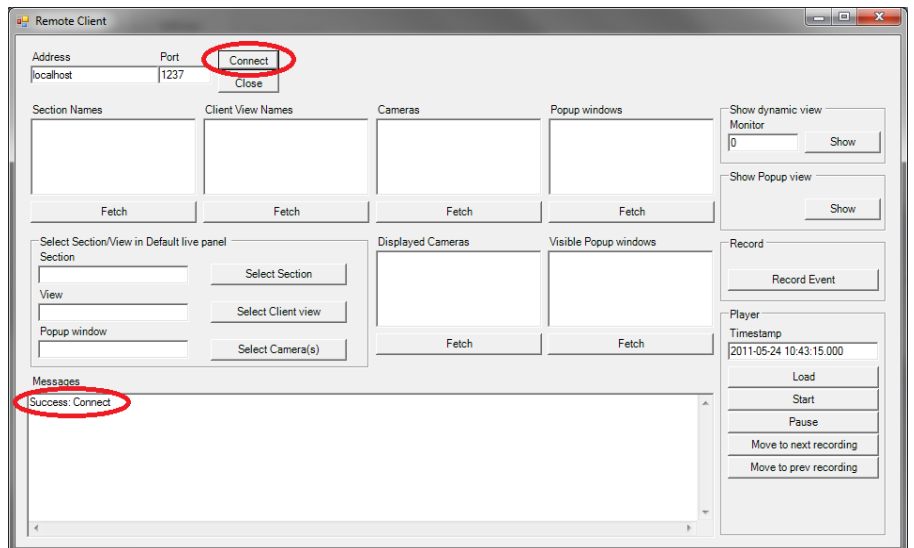


Figure 3.6 Connection to Ethisis Client successful.

Directly below the *Connect* and *Close* buttons are 6 lists; *Section Names*, *Client View Names*, *Cameras*, *Displayed Cameras*, *Popup windows*, and *Visible Popup windows*.

Below each list is a *Fetch* button. Click the *Fetch* button to retrieve the corresponding list.

*Section Names* – Clicking the *Fetch* button retrieves the current client configuration's list of *sections*.

*Client View Names* – Clicking the *Fetch* button retrieves the names of the views that are part of the currently selected *section* in Ethisis Client. If you first choose a section from the *Section Names* list in the sample application, you will get a list of view names of the views in the selected section in the sample application rather than the section that is selected in Ethisis Client.

*Cameras* – Clicking the *Fetch* button retrieves the names of all cameras part of the current client configuration. The list is the same as the list in the *Cameras* tool window in Ethisis Client.

*Displayed Cameras* – Clicking the *Fetch* button retrieves a list of names for all cameras currently displayed live in Ethisis Client.

*Popup windows* – Clicking the *Fetch* button retrieves a list of names for all popup windows defined in Ethisis Client.

*Visible Popup windows* – Clicking the *Fetch* button retrieves a list of names for all popup windows currently displayed in Ethisis Client.

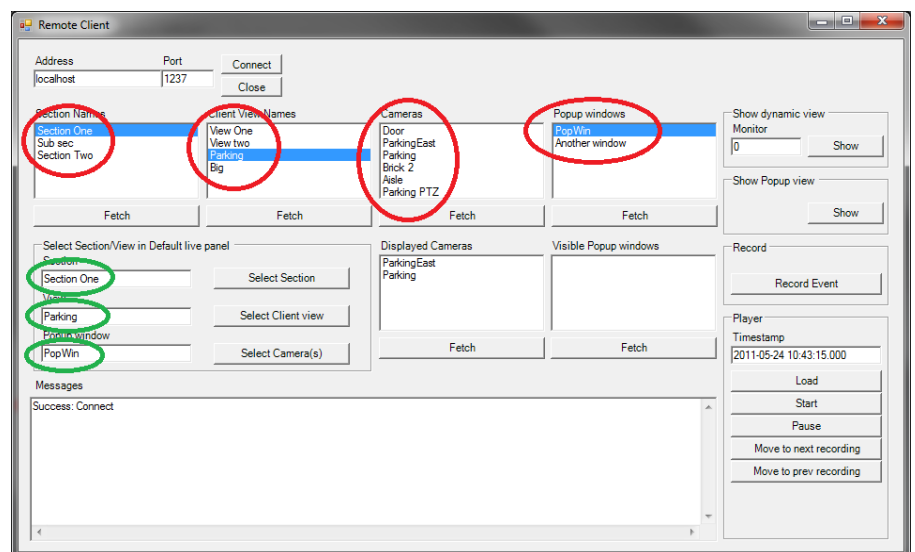


Figure 3.7 Sample application after clicking the *Fetch* buttons.

When you select a *section* by clicking its name in the *Section Names* list, the section name is copied to the *Section* field to the left of the *Select Section* button. The same goes for *Client View Names* and *Popup windows*. These fields are ringed in with green color in the example above.

In the example above, section *Sec one* and the view *Parking* are selected.

Click the *Select Section* button to change section in Ethisis Client and click the *Select Client view* button to change the view in Ethisis Client.

### Cameras list

The *Cameras* list behaves somewhat differently since you can select multiple cameras on the list. Click a camera name to select it. Click again to unselect it. By clicking several cameras, one at a time, you can have several cameras chosen in the list. See the example below.

Now, clicking the *Select Camera(s)* button will create a *dynamic view* in Ethisis Client and display the selected cameras in the *Default Live panel*.

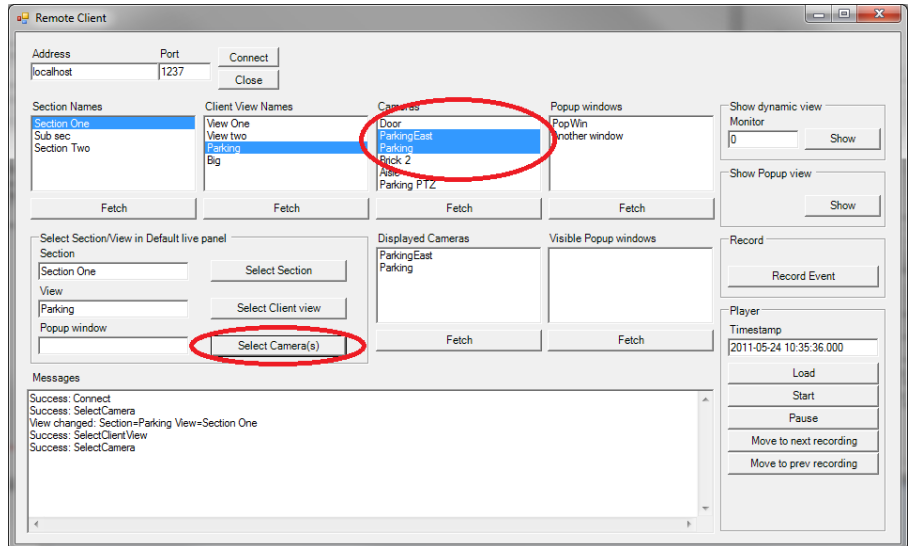


Figure 3.8 Two cameras selected.

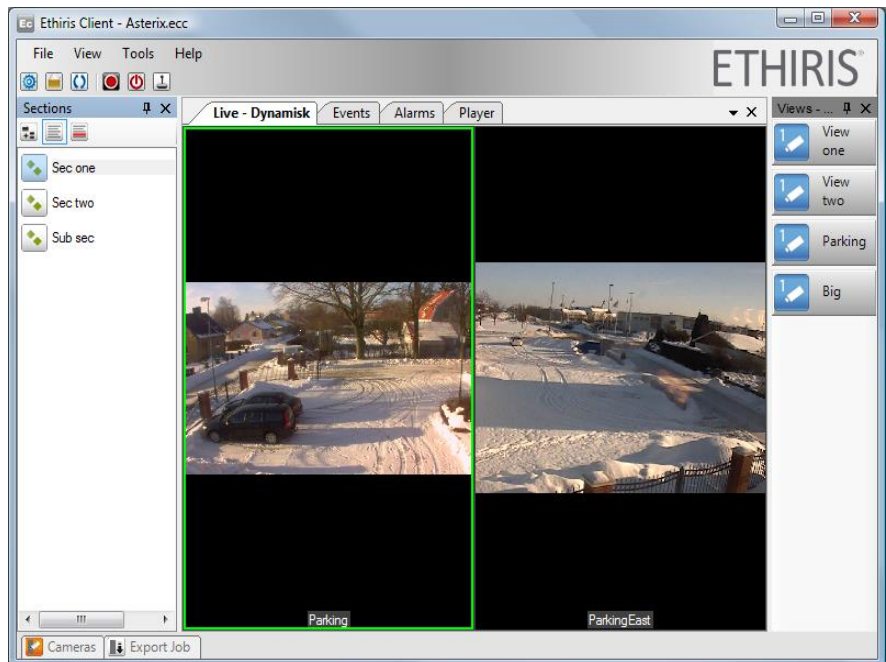


Figure 3.9 A dynamic view with the two chosen cameras in Ethis Client.

Clicking the *Select Camera(s)* button has the same effect as clicking the *Show* button with *Monitor 0* in the *Show dynamic view* section of the sample application.

You can enter another monitor number, such as *1* for monitor 1. Clicking the *Show* button then will create a dynamic view in its own window on the specified monitor.

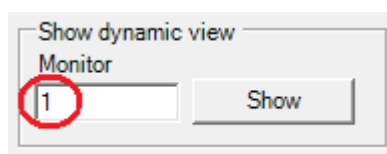


Figure 3.10 Show a dynamic view on monitor 1.

The dynamic view will display the cameras selected in the *Cameras* list.

### Popup windows

Click the *Show* button to show the preconfigured view specified under *View* in the preconfigured popup window specified under the *Popup window*. Furthermore, the cameras selected in list *Cameras* will populate undefined camera views in the view.

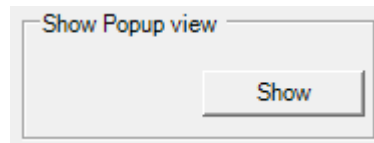


Figure 3.11 The Record Event button in the sample application.

There are a lot of possibilities with the function. If no view is selected, the cameras chosen in the list *Cameras* will be displayed in a dynamic view in the selected popup window. If no cameras are selected, the selected view will be shown in the selected popup window precisely as it is defined. Om neither view nor any cameras are selected, the specified popup window will be closed.

There is also a *Record* section in the sample application with a *Record Event* button.

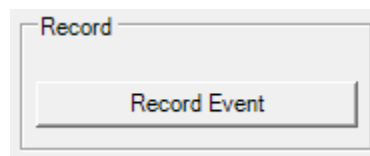


Figure 3.12 The Record Event button in the sample application.

Click the *Record Event* button to start event recording on the cameras that are selected in the *Cameras* list.

Finally, there is a *Player* section in the sample application.

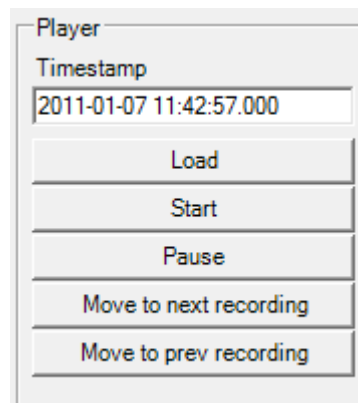


Figure 3.13 The Player section in the sample application.

When you click the *Load* button, Ethis Client brings up the *Player* tab, loads it with the cameras selected in the *Cameras* list, and sets the time ruler to the *Timestamp* in the *Timestamp* field in the sample application.

Now you can maneuver the Player with the other buttons in the sample application; *Start*, *Pause*, *Move to next recording*, and *Move to prev recording*.

Try them out!



*The sample application is not intended to be used directly in a real application!*

### WideQuick sample

If you selected the *WideQuick* sample in the installation, the sample project is located in a subfolder of the Ethisis installation folder. In general, this is *C:\Program Files\Kentima AB\Ethisis\Samples\RemoteClient\WideQuick*.

Note that the sample application only is supposed to demonstrate how to use *Remote control of Ethisis Client*. It is not intended to be used directly in a real application!

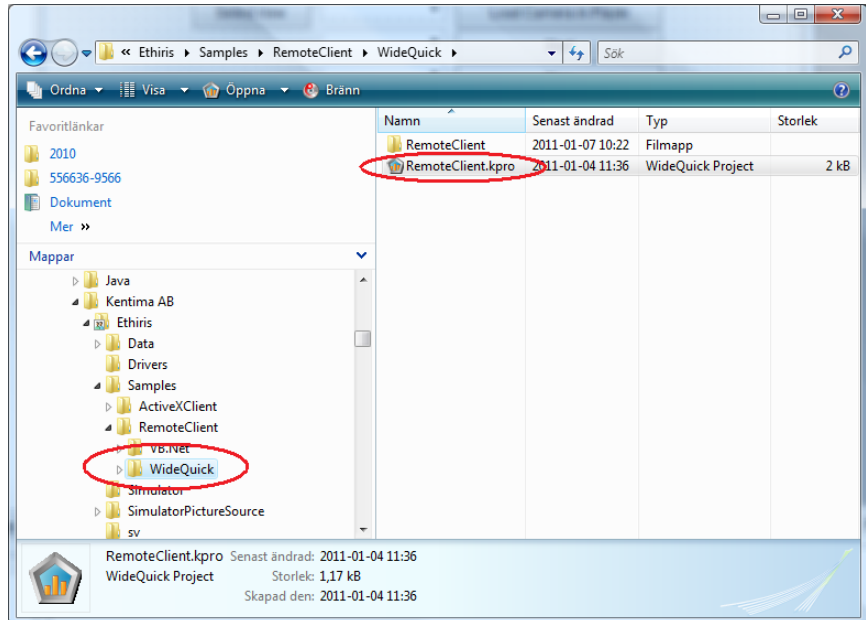


Figure 3.14 The Remote client sample project for WideQuick in Windows Explorer.

Just start WideQuick Designer and open the project *RemoteClient.kpro*.

When you run the project in preview, it looks similar to the VB.Net sample.

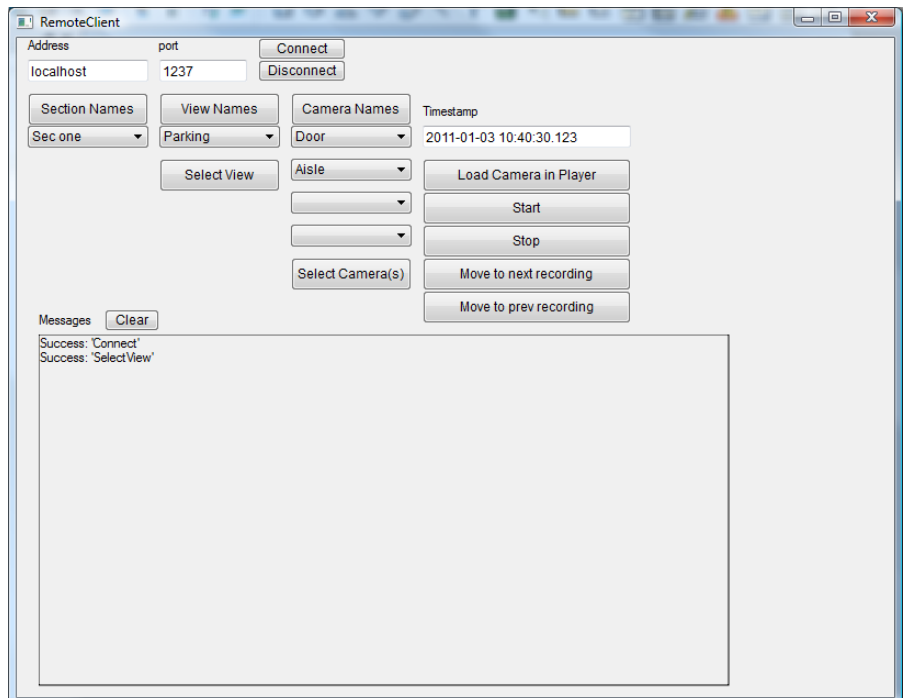


Figure 3.15 WideQuick preview for the sample project.

Most of the methods and events are implemented in this sample project. Please refer to the VB.Net example above for information about the purpose of the various buttons.

### 3.1.4 Remote control interface

When using the Ethisis Client Remote control, you use the interface of the control. The following is a reference to the various available properties, methods, events, and enumerations.

#### **Properties**

There are no properties for the Ethisis Client Remote control.

#### **Methods**

##### **Close()**

Closes the connection to Ethisis Client.

##### **Connect**(Address as String, Port as Long)

Connects to the Ethisis client instance to be controlled as specified by the parameters.

*Address* – A string containing the IP-address or the DNS name of the computer running the Ethisis Client.

*Port* – Port number that the Ethisis Client listens to for remote control commands. This is by default 1237.

##### **GetCameraNames()**

Initializes the transfer of an array of camera names for all cameras in the client. The result is returned in the *CameraNames* event.

##### **GetDisplayedCameras()**

Initializes the transfer of an array of camera names for all cameras that are currently displayed by the client. The result is returned in the *DisplayedCameras* event.

##### **GetPopupWindowNames()**

Initializes the transfer of an array of all popup windows in the client. The result is returned in the *PopupWindowNames* event.

##### **GetSectionNames()**

Initializes the transfer of an array of strings defining the currently defined section names in the Ethisis client. The result is returned in the *SectionNames* event.

##### **GetViewNames**([SectionName as String])

Initializes the transfer of an array of client view names in the current or specified section. The result is returned in the *ClientViewNames* event. The parameter is optional. If specified, the client views belonging to *SectionName* are returned. If no section is specified, the client views belonging to the currently selected section in Ethis Client are returned.

#### **GetVisiblePopupWindows()**

Initializes the transfer of an array of popup window names that are currently visible in the client. The result is returned in the *VisiblePopupWindows* event.

#### **LoadCamerasInPlayerByTimestamp**(Timestamp as String, CameraNames[] as String)

Load the recorded video for the specified camera(s) into the client video player.

*Timestamp* – A string containing the desired timestamp for the Player. This should be in the form *YYYY-MM-DD hh:mm:ss.ttt*, e.g. *2011-01-12 14:14:53.000*.

*CameraNames[]* – An array of strings containing the names of the desired cameras to load in the Player.

#### **MoveToNextRecording()**

Move player to beginning of next recording.

#### **MoveToPrevRecording()**

Move player to the beginning of the previous recording.

#### **PausePlayer()**

Pause the client video player.

#### **RecordEvent**(CameraNames[] as String)

Triggers recording of an event sequence for the specified camera(s).

*CameraNames[]* – An array of strings containing the names of the desired cameras to start event recording for.

#### **ReloadConfiguration**([ForceReload as Boolean])

Instructs the client to initiate a reload of its configuration. The parameter is optional. If *ForceReload* is *true*, the configuration will always be reloaded. If the parameter is left out or is *false*, the configuration will only be reloaded if the configuration file has actually been updated since the configuration was last loaded by the client. Also, look at the event *ReloadPending*.

#### **SelectCamera**(CameraNames[] as String)

Displays a dynamically composed view of the listed cameras in the Default Live panel in Ethis Client.



*CameraNames[]* – An array of strings containing the names of the desired cameras to create a dynamic view for.

**SelectSection**(SectionName as String, [ViewName as String])

Selects a specified section in the Ethisis client. If *ViewName* is specified, the view is selected.

*SectionName* – A string containing the name of the desired Section to select.

*ViewName* – An optional string containing the name of the desired View to select.

**SelectView**(ViewName as String, [SectionName as String])

Select a specific client view in the current or specified section. The parameter *SectionName* is optional. If specified, a client view belonging to *SectionName* is selected. If no section is specified, the specified client view belonging to the currently selected section in Ethisis Client is selected.

*ViewName* – A string containing the name of the desired View to select.

*SectionName* – An optional string containing the name of the desired Section to select.

**ShowDynamicView**(Monitor as Long, CameraNames[] as String)

Displays a dynamically composed view of the listed cameras on the specified monitor.

*Monitor* – A long value specifying the monitor where the dynamic view will be created. A value of 0 will display the cameras in the *Default Live panel*. A value of 1 will create a new dynamic window on monitor 1.

*CameraNames[]* – An array of strings containing the names of the desired cameras to create a dynamic view for. An empty array will close the dynamic view on the specified monitor.

**ShowPopupView**(PopupWindowName As String, ViewName as String, CameraNames() As String)

Displays the preconfigured view specified in parameter *ViewName* in the preconfigured popup window specified in parameter *PopupWindowName*. It also populates undefined camera views with cameras specified in parameter *CameraNames*.

*PopupWindowName* – A value of type *String* specifying the popup window to use.

*ViewName* – A value of type *String* specifying the view to use as layout to show the cameras in parameter *CameraNames()*. If this parameter is empty, the cameras will be displayed in a dynamically created view. If the view contains defined camera views (cameras, rounds, hotspots, or background images), these will not be substituted with cameras specified in parameter *CameraNames*. Only camera views of type *Camera*, but without a defined camera, will be replaced.

*CameraNames[]* – An array of strings containing the names of the desired cameras the selected or dynamic view will contain. An empty array will show the defined view as it is defined.

If both parameters *ViewName* and *CameraNames* are empty, the specified popup window will close.

#### **StartPlayer()**

Start playing the video that is loaded into the client video player.

## Events

### CameraNames(Names[] as String)

This event is sent from the remote control after the command *GetCameraNames* has been sent to the control. The following parameters are sent in the event:

*Names[]* – An array of strings containing the names of the available cameras in Ethisis Client.

### DisplayedCameras(Cameras[] as String)

This event is sent from the remote control after the command *GetDisplayedCameras* has been sent to the control. The event is also sent spontaneously from the remote control when any change in which cameras are currently shown in Ethisis Client. The following parameters are sent in the event:

*Cameras[]* – An array of strings containing the names of the cameras currently displayed in Ethisis Client.

### Failure(Command as eCmdCodes, Code as Long, Message as String)

This event is sent from the remote control when a failure is detected in an operation that is not directly initiated by a call from the hosting application. The following parameters are sent in the event:

*Command* – An integer indicating which operation went wrong. For more information, see *Enumerations* below

*Code* – If applicable, this parameter contains an error code.

*Message* – A string containing information about the problem.

### PlayerLoaded(Names[] as String, Timestamp as String)

This event is sent from the remote control whenever the Player in Ethisis is loaded. The following parameters are sent in the event:

*Names[]* – An array of strings containing the names of the cameras loaded in the Player.

*Timestamp* – A string with the timestamp in the time ruler in the Player when it was loaded.

### PlayerState(State as ePlayerState)

This event is sent from the remote control when there is a change in the Player state, e.g., when the Player starts or pauses playback of the video. The following parameter is sent in the event:

*State* – There are a total of 3 different states as described below in the *Enumerations* section. The *State* parameter describes the new state of the Player.

### **PopupWindowNames**(Names[] as String)

This event is sent from the remote control after the command *GetPopupWindowNames* has been sent to the control. The following parameters are sent in the event:

*Names[]* – An array of strings containing the names of the available popup windows in Ethisis Client.

### **Recording**(CameraName as String, Start as Boolean)

This event is sent from the remote control when a camera starts or stops recording. The following parameter is sent in the event:

*CameraName* – The name of the camera that starts or stops recording.

*Start* – A Boolean value that is *true* when recording starts and *false* when recording stops.

### **ReloadPending**(WillReload as Boolean)

This event is sent from the remote control as a response to the command *ReloadConfiguration(...)*. The following parameter is sent in the event:

*WillReload* – A Boolean value that is *true* when the client will reload its configuration or *false* if the client will not reload its configuration.

### **SectionNames**(Names[] as String)

This event is sent from the remote control after the command *GetSectionNames* has been sent to the control. The following parameters are sent in the event:

*Names[]* – An array of strings containing the names of the available sections in Ethisis Client.

### **Success**(CommandAs eCmdCodes)

This event is sent from the remote control when a command was successful. The following parameters are sent in the event:

*Command* – An integer indicating was successful. For more information, see *Enumerations* below

### **ViewChanged**(Section as String, View as String)

This event is sent from the remote control when the current *View* in Ethisis Client is changed. The following parameter is sent in the event:

*Section* – The name of the possible new current *section*.

*View* – The name of the new current *view*.

### **ViewNames**(SectionName as String, Names[] as String)

This event is sent from the remote control after the command *GetViewNames* has been sent to the control. The following parameters are sent in the event:

*SectionName* – The name of the current or specified *section*.

*Names[]* – An array of strings containing the names of the available *views* in the section.

#### **VisiblePopupWindows(PopupWindows[] as String)**

This event is sent from the remote control after the command *GetVisiblePopupWindows* has been sent to the control. The event is also sent spontaneously from the remote control when any popup window is opened or closed. The following parameter is sent in the event:

*PopupWindows[]* – An array of strings containing the names of the popup windows currently visible in Ethisis Client.

#### **Enumerations**

**eCmdCodes** contains 13 different values from 0 to 12 as described below:

*eccUnknown* – 0. Command *Unknown*.

*eccConnect* – 1. Command *Connect*.

*eccDisconnect* – 2. Command *Close*.

*eccSelectSection* – 3. Command *SelectSection*.

*eccSelectView* – 4. Command *SelectView*.

*eccSelectCamera* – 5. Command *SelectCamera*.

*eccRecordEvent* – 6. Command *RecordEvent*.

*eccLoadCamerasInPlayerByTimestamp* – 7. Command *LoadCamerasInPlayerByTimestamp*.

*eccStartPlayer* – 8. Command *StartPlayer*.

*eccPausePlayer* – 9. Command *PausePlayer*.

*eccShowDynamicView* – 10. Command *ShowDynamicView*.

*eccMoveToNextRecording* – 11. Command *MoveToNextRecording*.

*eccMoveToPrevRecording* – 12. Command *MoveToPrevRecording*.

*eccShowPopupView* – 13. Command *ShowpopupView*.

**ePlayerState** contains 3 different values from 0 to 3, as described below:

*epsUnknown* – 0. The Player is in an unknown state.

*epsRun* – 2. The Player is in state *Started*.

*epsPause* – 3. The Player is in state *Pause*.



<b>4 Modbus OPC Server</b>	<b>4:1</b>
4.1 Modbus OPC Server .....	4:1
4.1.1 Overview.....	4:1
4.1.2 Installation.....	4:1
4.1.3 Configuration.....	4:1





# 4 Modbus OPC Server

## 4.1 Modbus OPC Server

### 4.1.1 Overview

The purpose of the Kentima Modbus OPC Server is to communicate with systems using the Modbus protocol. In this section, we will set up communication between the Ethisis Server and a Beckhoff BK9000 module, used for distributed I/O.

To get things working, we need at least an *Extended* license level for the Ethisis Server, since we will use the OPC Client functionality in Ethisis Server. We also need the *Kentima Modbus OPC Server* installed on the same computer as the Ethisis Server. Then the Ethisis Server will use COM rather than DCOM, which is much harder to set up due to complex security settings. DCOM is beyond the scope of this manual.

### 4.1.2 Installation

Run the Kentima Modbus OPC Server setup on the same computer as the Ethisis Server runs on. This setup program requires a *Product code*. Make sure you have access to a product code for Kentima Modbus OPC Server before you install the OPC Server.

### 4.1.3 Configuration

First, we have to configure the OPC Server. **Start** the *Modbus OPC Server* program on the Start menu.

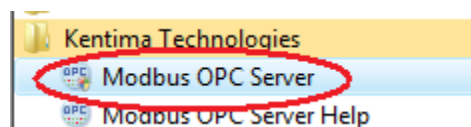


Figure 4.1 The Modbus OPC Server program on the Start menu.

**Right-click** on *KentimaOPCModbus* in the treeview and **select** *Create TCP connection* in the popup menu.

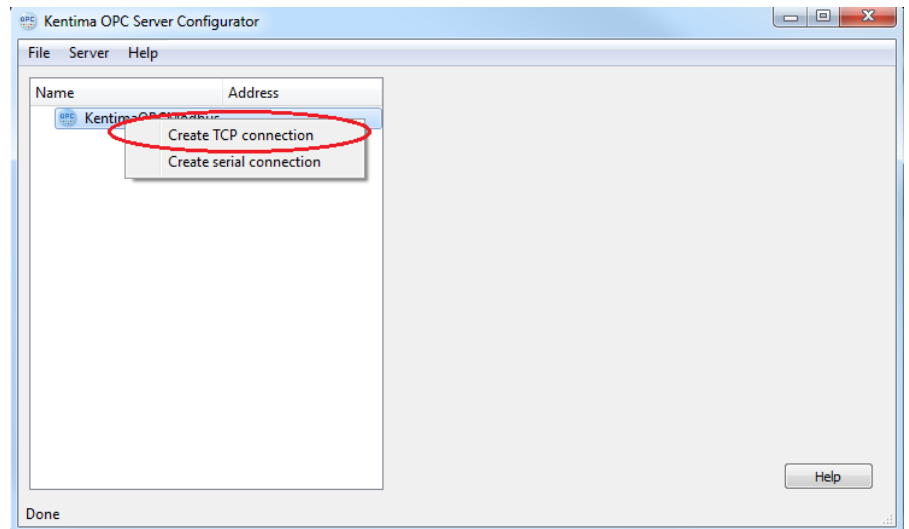


Figure 4.2 Create a TCP connection.

A new node called *Default TCP Connection* is created in the treeview. **Enter** the IP address of the Beckhoff device.

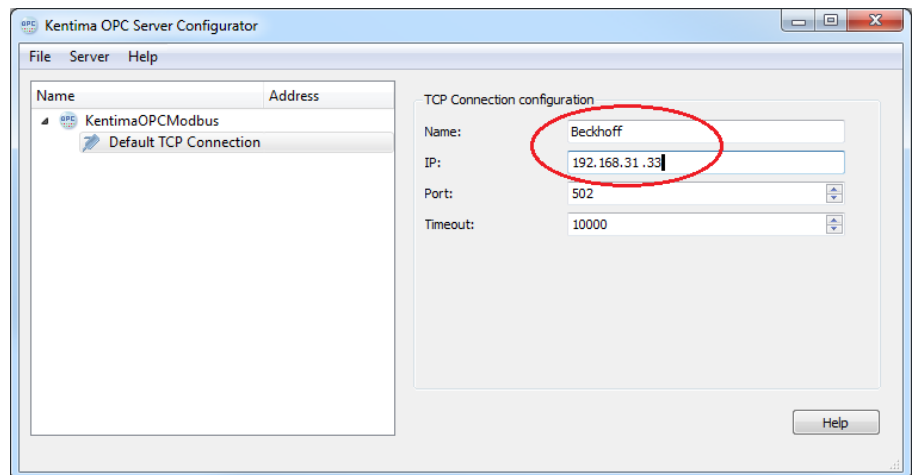


Figure 4.3 Enter IP address.

If you want, you can change the name of the TCP connection, but it is not necessary. I will change it to *Beckhoff* for clarity.

**Right-click** the *Default TCP Connection (Beckhoff)* node in the treeview and **select** *Create slave* in the popup menu. Note that the name is updated in the treeview when you click the node.

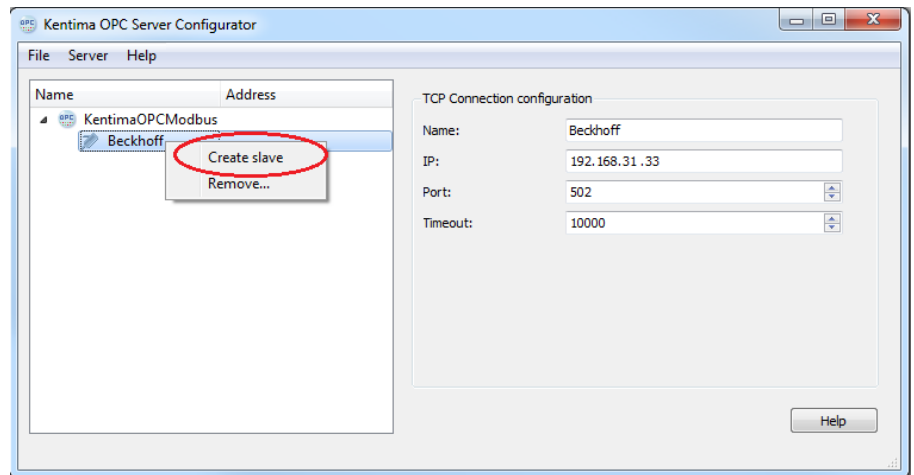


Figure 4.4 Create a slave.

A new node appears in the treeview, *Default Slave*. You can change the name for this node also. In this example, we choose the name *Fence*, pretending that the device is used for some kind of detector for a fence.

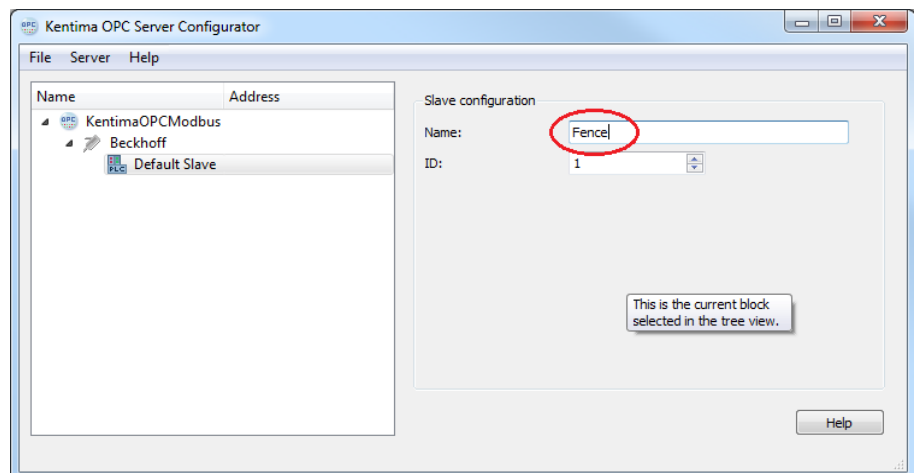


Figure 4.5 Slave created.

In this example, we have a digital input card (KL1104) connected to the Beckhoff device. There are several different models of input cards with various numbers of inputs. This particular card has four digital inputs.

The next step is to configure the digital inputs in the OPC Server. All communication is conducted through messages. So, first, we create an input message. **Right-click** the *Default Slave (Fence)* node and **select** *Add read input* in the popup menu.

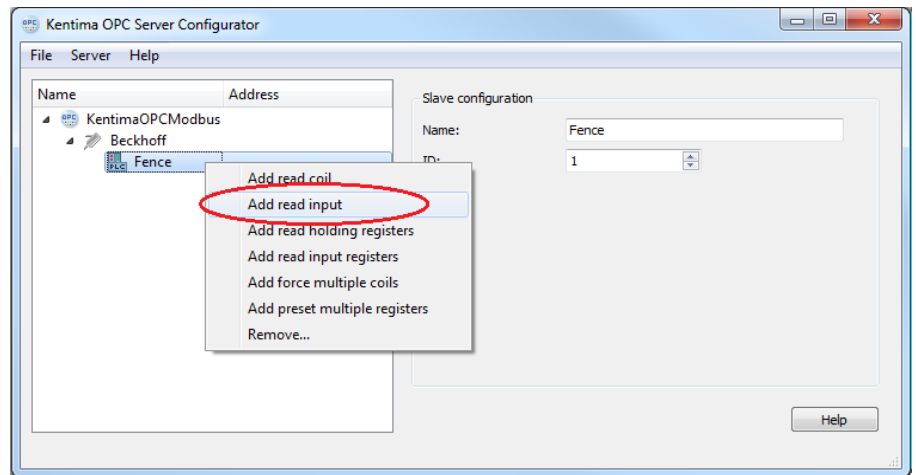


Figure 4.6 Add digital input message.

Another node appears, *Default Message 0-0*. Change the name of the message to something meaningful, like *Inputs*.

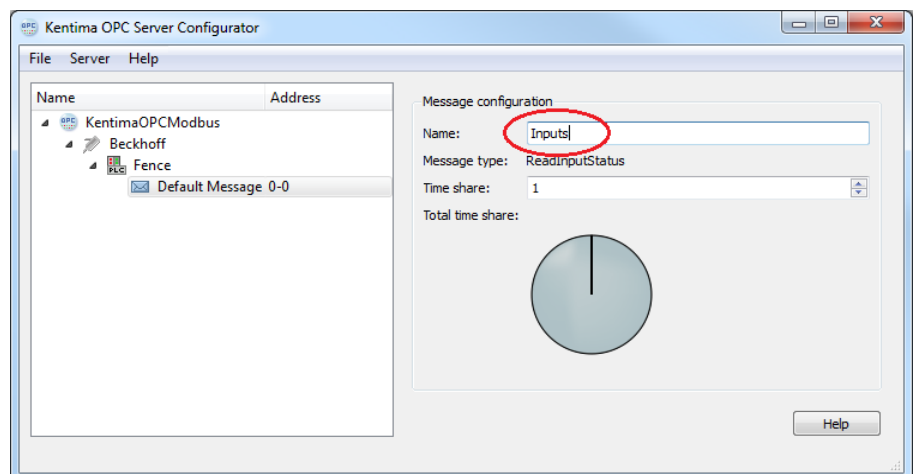


Figure 4.7 Read input message added.

Now it's time for the inputs. **Right-click** the new node *Default Message 0-0* (*Inputs 0-0*) and **select** *Create variable* in the popup menu.

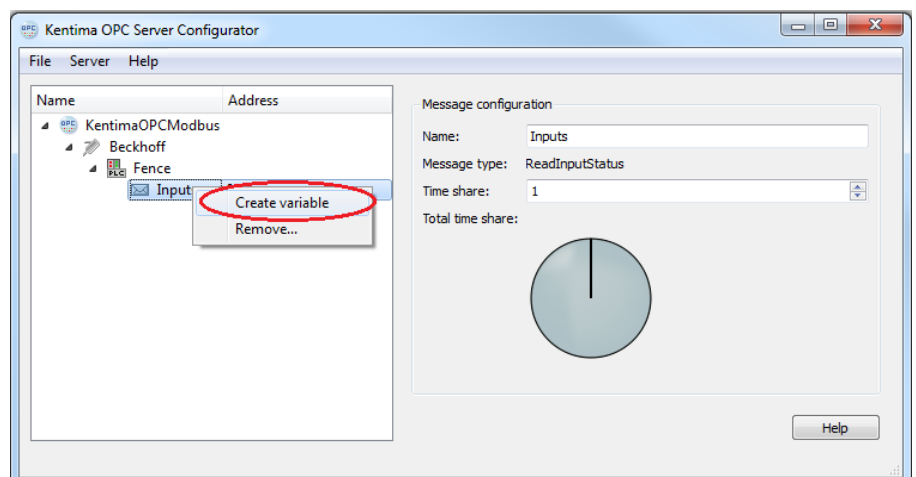


Figure 4.8 Create variable.

A new node appears. Change the name to something more sensible, like *Input1*. The address is as default *0*. This is the first address on the device.

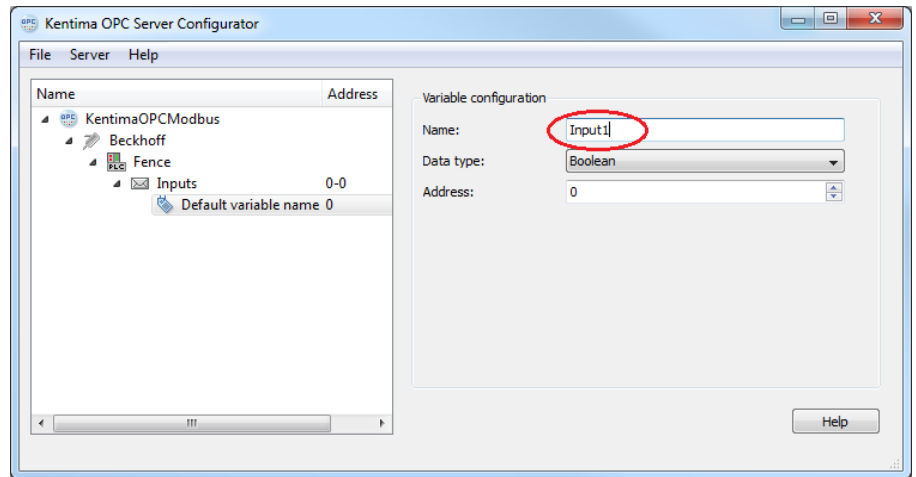


Figure 4.9 Rename the new variable.

As mentioned before, we have four digital inputs. You can use the *Duplicate* menu to add three more digital inputs to the configuration. Now the address is automatically increased, and the name automatically generated for the following inputs.

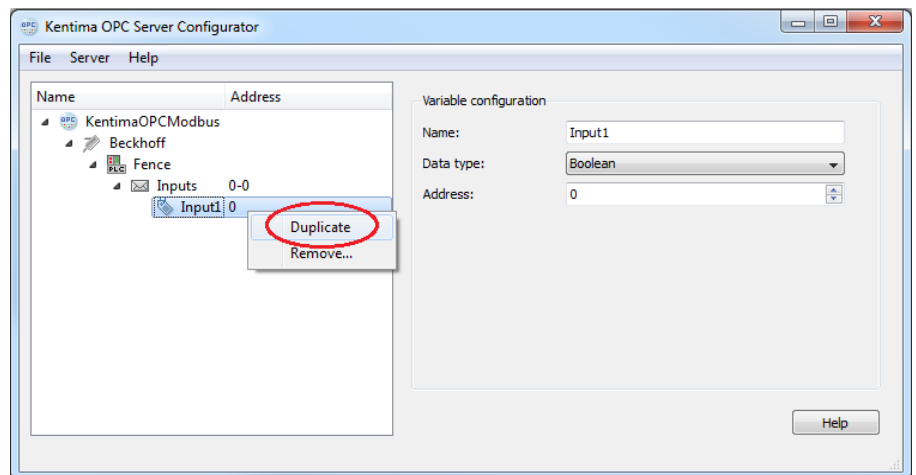


Figure 4.10 Duplicate variables.

After we have done this three times, we have four digital inputs.

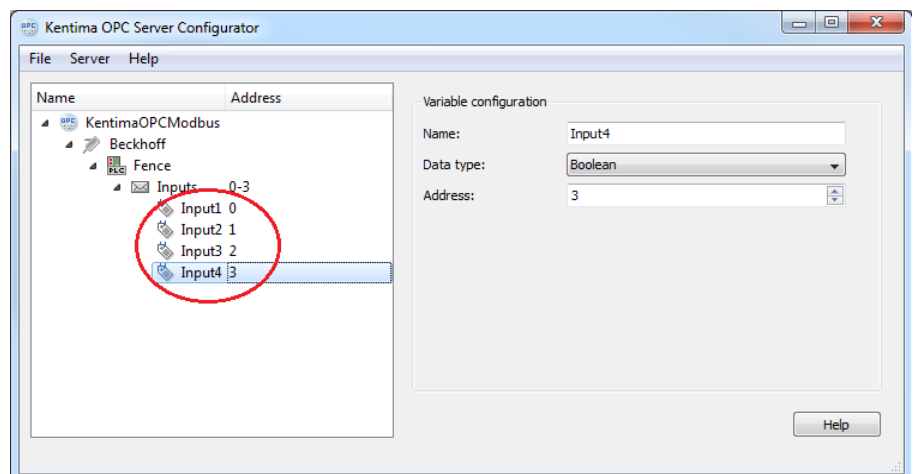


Figure 4.11 4 digital inputs created.

We are now finished configuring our digital input signals. Let's configure four digital output signals as well. Just as for the Beckhoff input cards, there are several different types to choose from. In this example, we use a card with four digital outputs, called *KL2134*.

Since we already have configured some aspects of the device, like the IP address, we can go directly on to the output message.

**Right-click** the *Fence* node in the tree view and **select** the *Add force multiple coils* menu item.

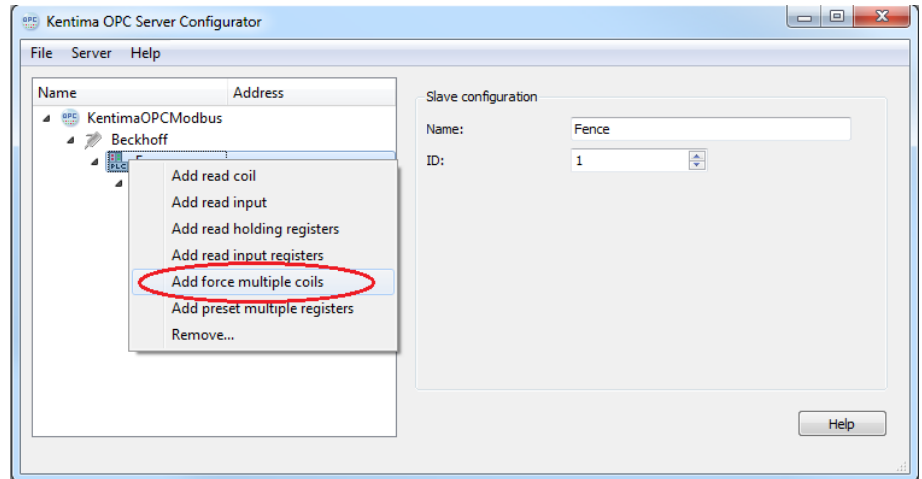


Figure 4.12 Add a message for output signals.

A new message is created.

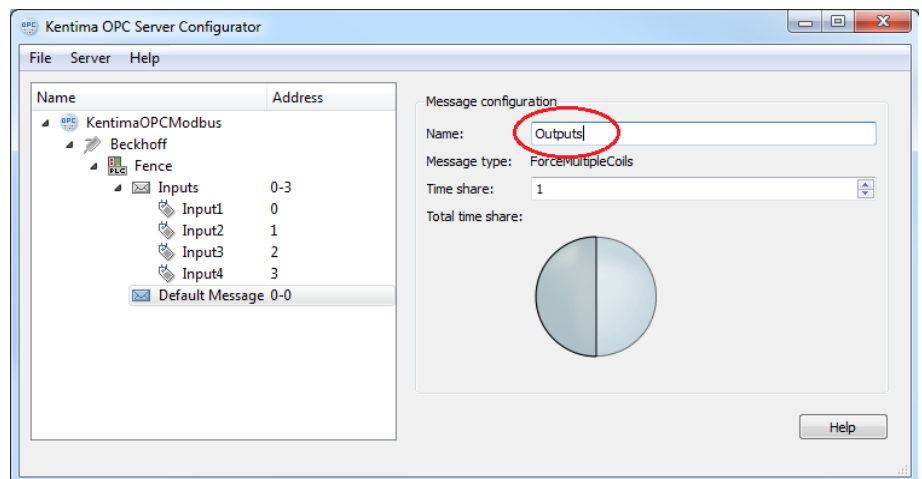


Figure 4.13 Created Output message.

**Change** the name of the message to, e.g., *Outputs*. The change has no effect until you click in the treeview.

The next step is to create output signals.

**Right-click** the *Outputs* node and **select** *Create variable*.

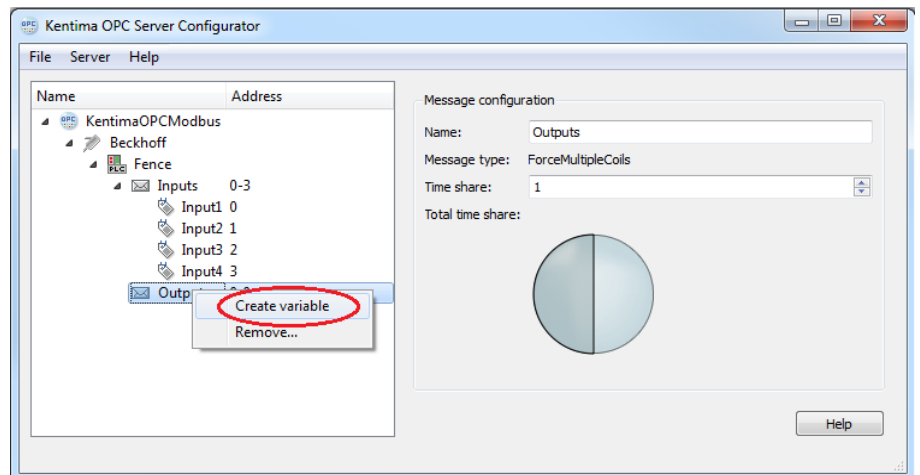


Figure 4.14 Create Output variable.

A new variable is created. **Select it** in the tree view and **rename** it to *Output1*.

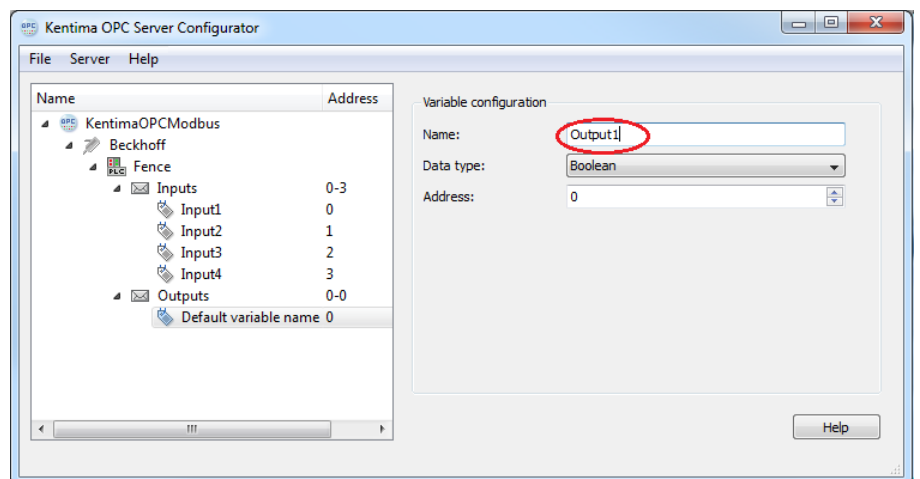


Figure 4.15 New Output variable created.

**Note!** In this example, we are using a *Beckhoff* device. The first address for output signals is 0. If you are using an *ADAM* device, the first output address starts at 16.

Duplicate the output variable three times.

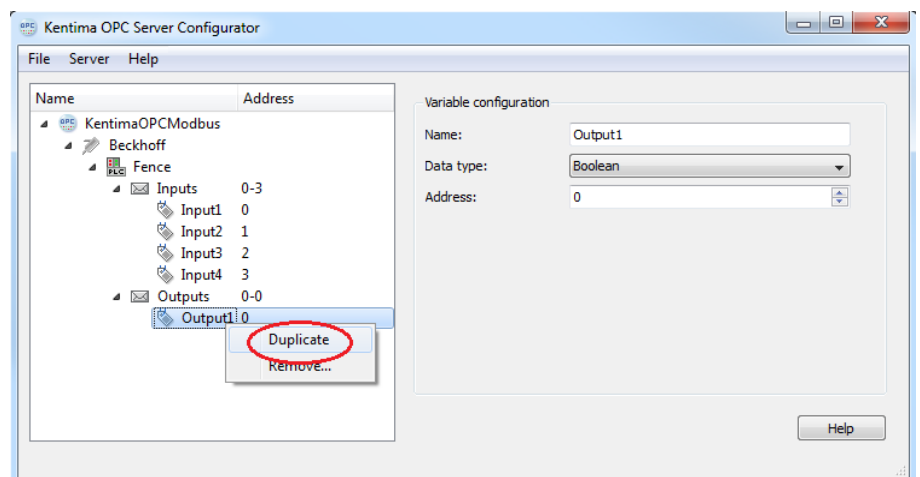


Figure 4.16 Duplicate the Output variable three times.

When you are done, the configuration dialog should look like this:

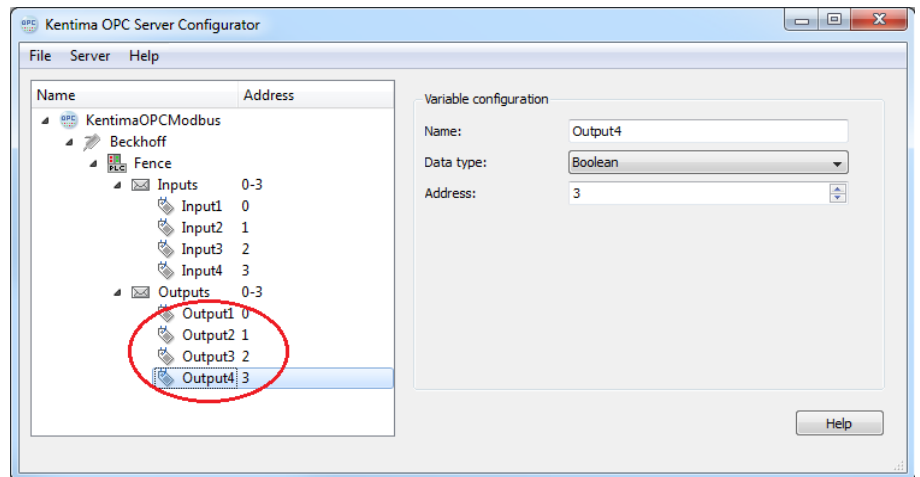


Figure 4.17 All messages and variables created.

Alright, now we have configured the OPC Server for TCP communication with a device with a specific IP address. We have created one message for four digital inputs and one message for four digital outputs, which will be communicated by the OPC Server.

The next step is to save the configuration and tell the OPC Server to use the new configuration.

Select the menu *Server->Set active config file....*

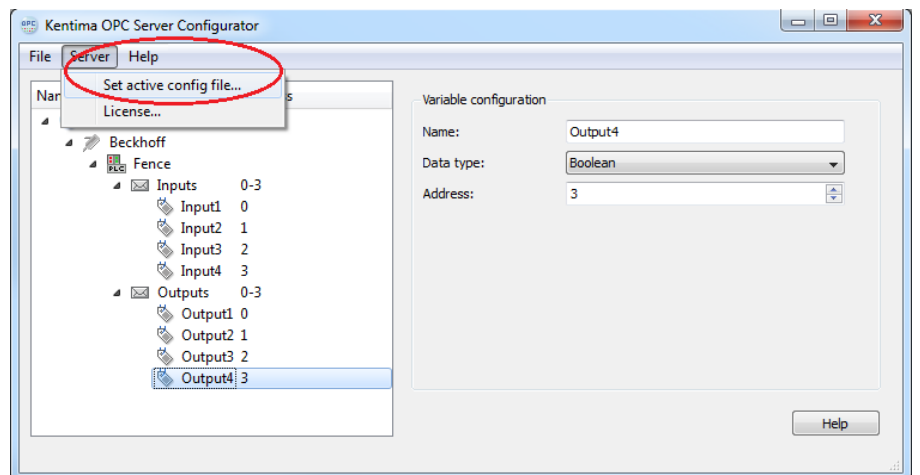


Figure 4.18 Activate configuration.

Click *Save* in the dialog that appears.



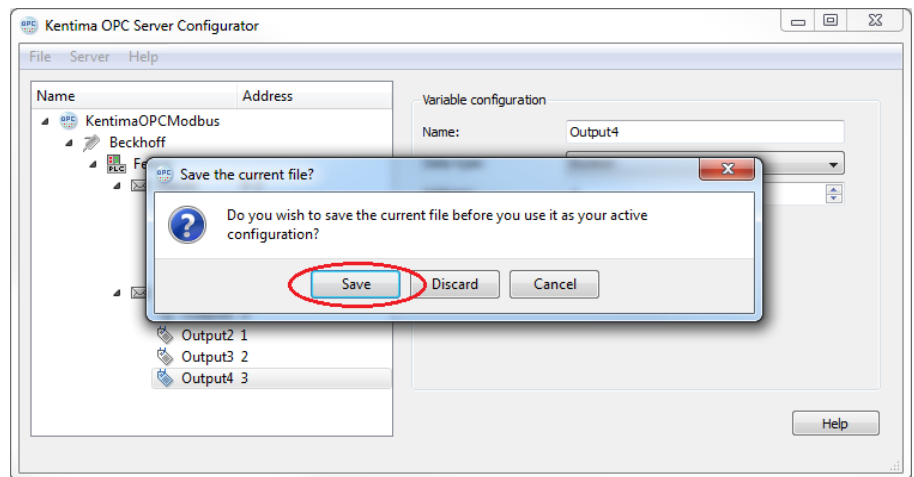


Figure 4.19 Save the new configuration.

A message appears informing that the OPC Server was restarted. It is restarted to read the new configuration.

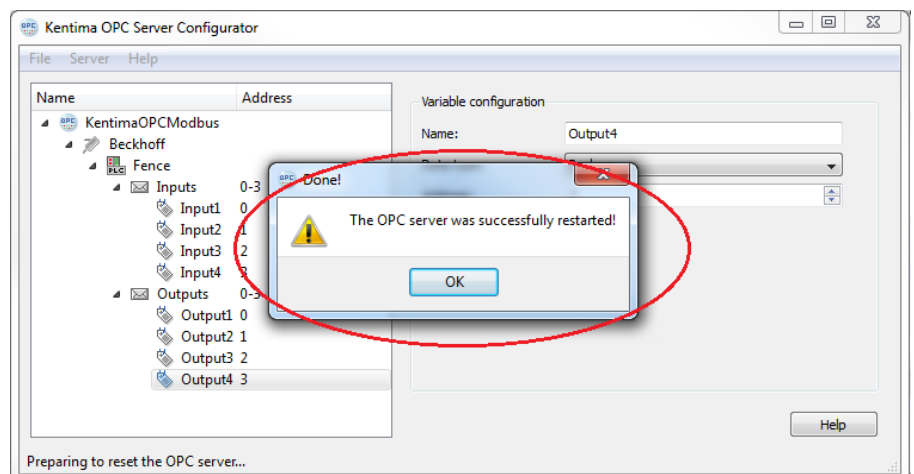


Figure 4.20 OPC Server restarted.

Now we are done with the OPC Server configuration.

The next step is to configure Ethisis to communicate with the OPC Server.

**Start Ethisis Admin** and **double-click** the *OPC Servers* node in the tree view.

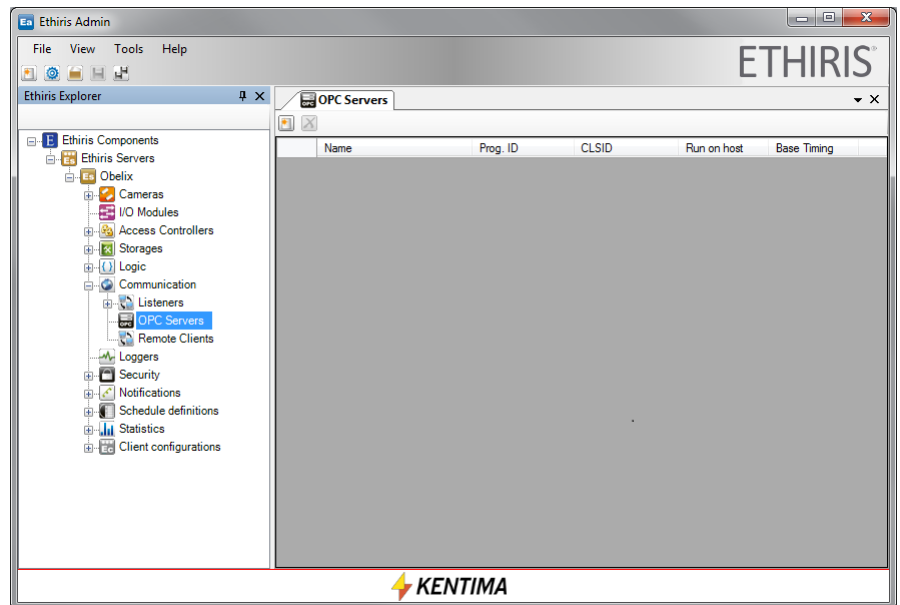


Figure 4.21 OPC Servers panel in Ethisis Admin.

Create a new OPC Server by clicking the *New OPC Server* button.

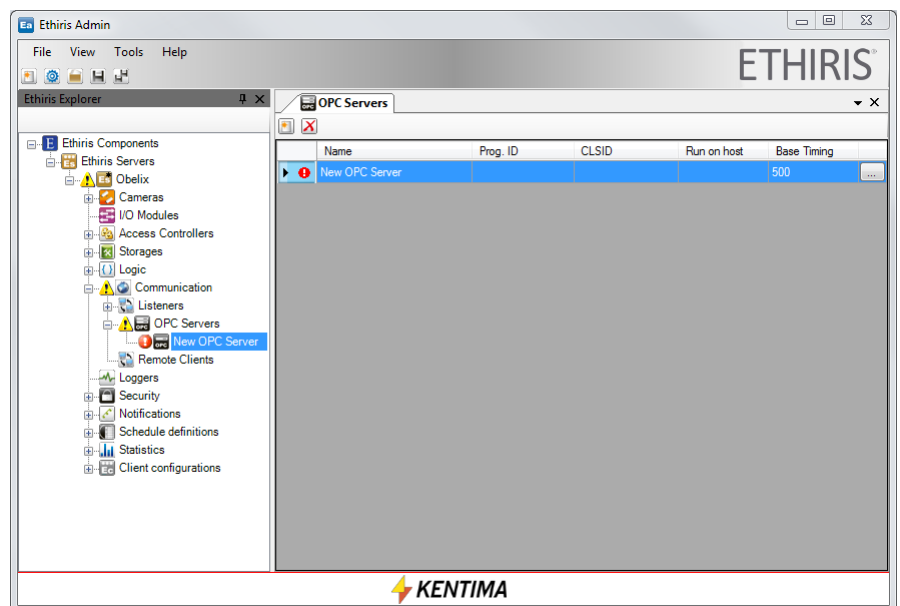


Figure 4.22 New OPC Server added.

Change the name to something else, e.g., *Beckhoff*.

**Click** the *Browse* button to the far right for selecting an OPC Server. In the browse dialog, **select** the *Kentima OPC server for Modbus*.

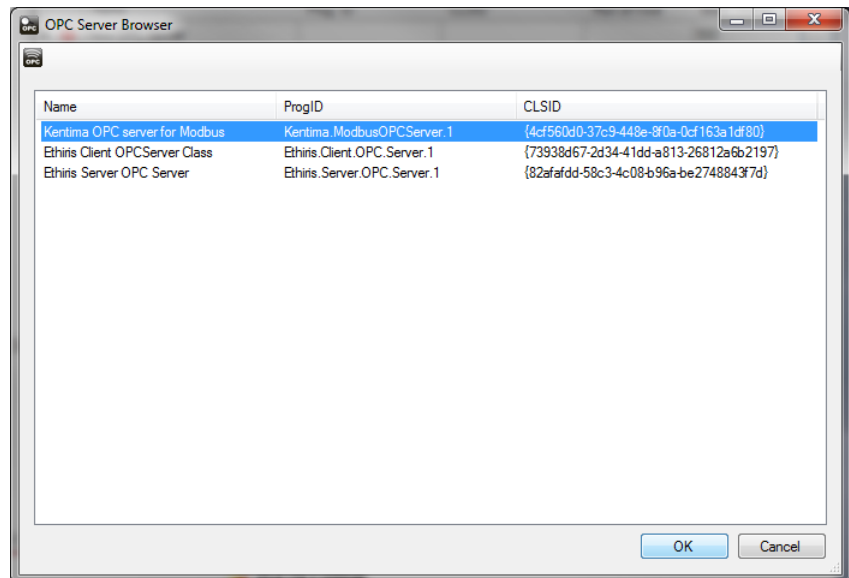


Figure 4.23 Select Kentima OPC server for Modbus

**Right-click** the *new node* in the tree view and **select** *New->OPC Group* in the popup menu.

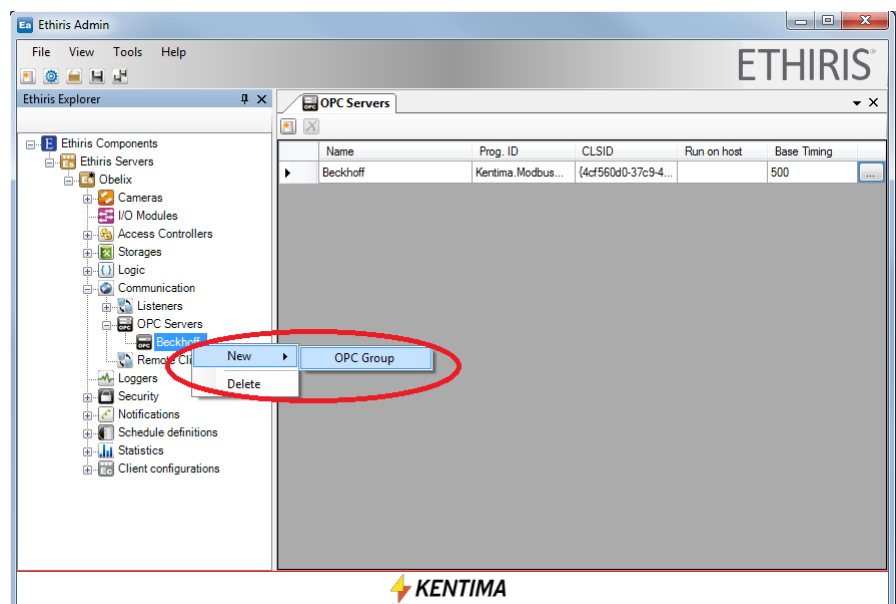


Figure 4.24 Create a new OPC Group.

A new node is created in the tree view.

**Double-click** the new *node* to open the corresponding panel.

**Change** the *name* of the group to something meaningful, e.g., *Inputs*.

To the right in the panel, there is an *OPC Item Browser* tool window that is docked. You can browse the OPC Server by expanding the treeview in the upper pane. You have to select each node before you can expand it.

**Select** the node *Inputs* in the upper pane. Now, the four input signals are visible in the lower pane.

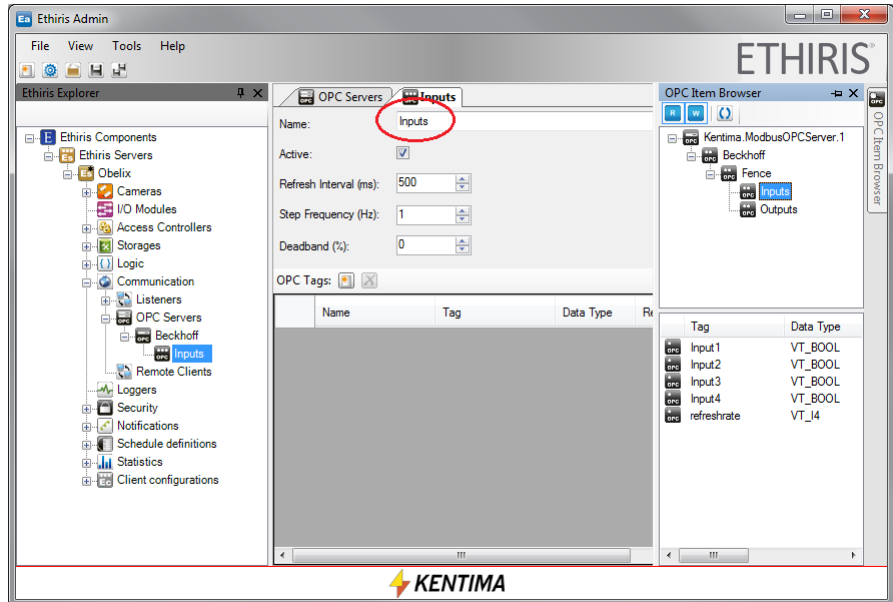


Figure 4.25 Rename the new OPC Group.

Select the four input signals in the lower pane and drag-and-drop them into the OPC-Tags list to the left.

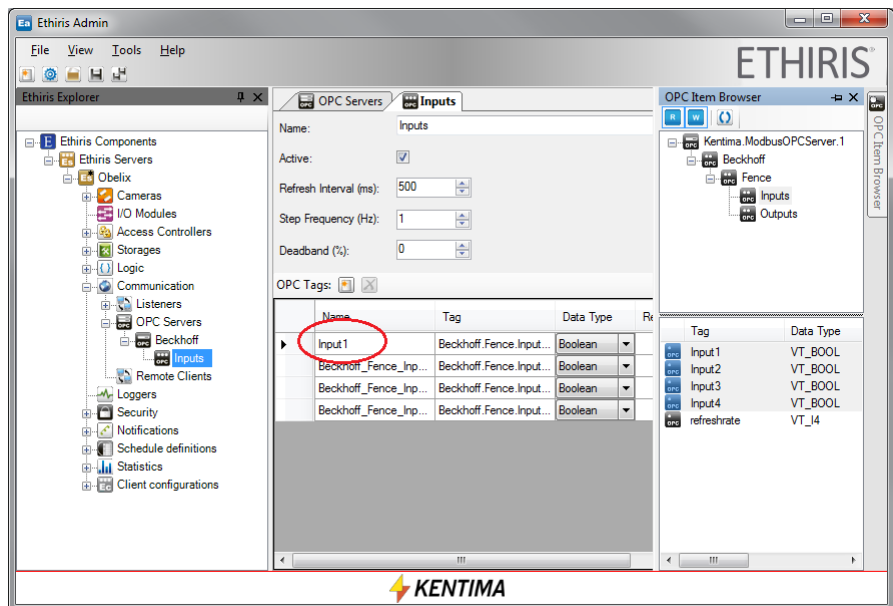


Figure 4.26 OPC tags added.

When the tags are added to the list, you can rename them to a somewhat shorter name.

If you like, you can create another group for the output signals and add the four output signals to that group. An alternative is to add all eight signals (four input and four output) to the same OPC Group.

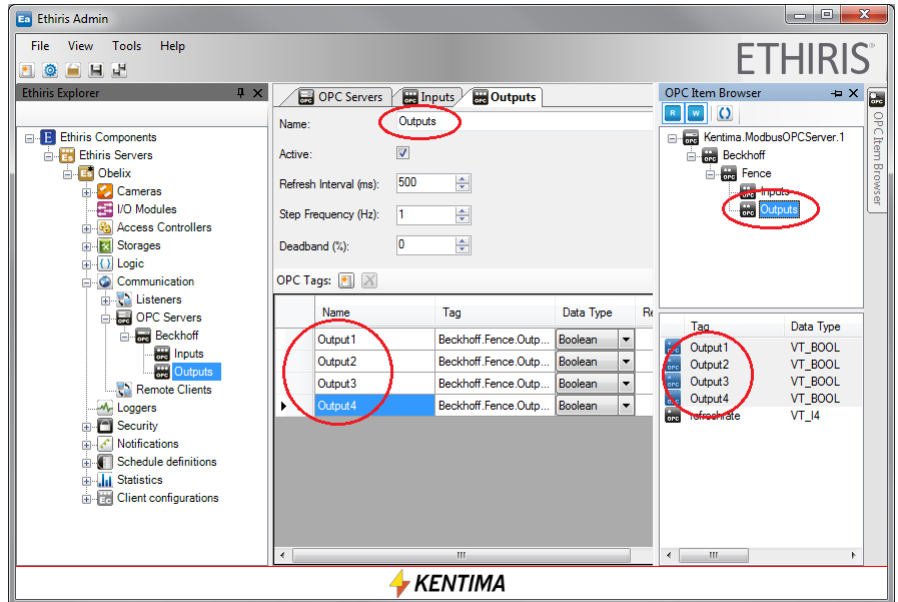


Figure 4.27 OPC tags for the four output signals also added.

Now, the new signals are available in Ethisis Server's data store via the *Variable Browser* or via Ethisis Client for presentation in live.

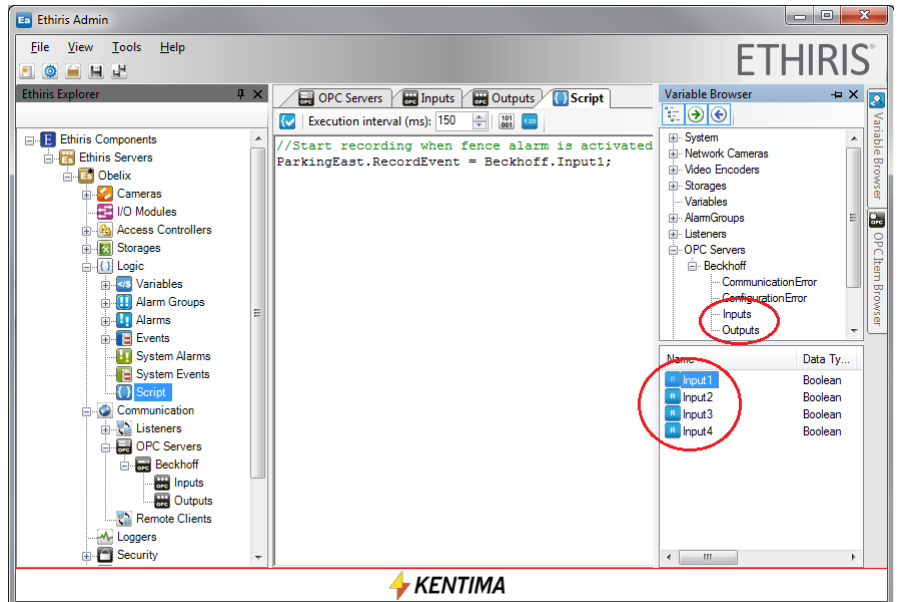


Figure 4.28 New signals available in Variable Browser.



## 5 Listeners

5:1

5.1 Listeners.....	5:1
5.1.1 Overview.....	5:1
5.1.2 Listeners general .....	5:2
5.2 Types of listeners .....	5:3
5.2.1 TCP-inbound .....	5:3
5.2.2 TCP-outbound.....	5:4





# 5 Listeners

## 5.1 Listeners

### 5.1.1 Overview

Listeners are first of all used for listening to messages from external equipment such as e.g., cameras and video encoders. But, somewhat contradictable, listeners can also be used for sending information to external equipment, and in that way, you can achieve two-way communication. The primary purpose, though, is to receive information, hence the name *Listeners*.

#### **Licensing**

To have all main types of listeners working, at least license level *Extended* is needed for Ethisis Server.

There is one listener with limited functionality that is available for all license levels. This listener is automatically created and cannot be deleted. The name of the listener is *TCP*. For more information, see the manual *Admin - Configuration for Ethisis*.

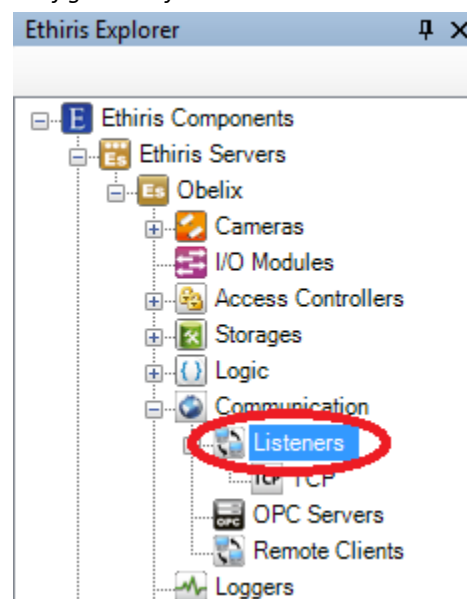


Figure 5.1 The node Listeners in Ethisis Explorer treeview.

### ***Listener types***

There are four main types of listeners:

- TCP inbound
- TCP outbound
- HTTP inbound
- HTTP outbound

Which type to use depends on in which way the external equipment can communicate.

## **5.1.2 Listeners general**

Listeners represent a relatively complex part of Ethisis. The upside is that they can be utilized in many situations for communicating with external equipment.

Which protocol to use is determined by *Type*, where you can choose between *TCP* and *HTTP*.

Via *Type*, you also select which one of Ethisis Server or the external equipment is responsible for starting the communication.

*Inbound* designates that it is the external equipment that is supposed to take the initiative of the communication. In this case, Ethisis Server listens to the specified port and waits for the external equipment to start sending messages. Note that the specified port has to be open in a possible firewall.

*Outbound* designates that it is Ethisis Server that is supposed to take the initiative of communication. Then Ethisis Server will actively try to connect to the external equipment using the specified IP address and port.

When a connection is established, and the communication commences Ethisis Server receives a text from the external equipment (regardless of whether the type is inbound or outbound). Now the whole idea is to activate *triggers by matching* corresponding texts in the data received by Ethisis Server.

A *trigger* consists of three parts; *Name*, *Function*, and *Match (Text to match)*.

Each trigger is represented in the Ethisis Server data store as a variable of type *Boolean*. A Boolean variable can hold two different values; *true* or *false*. In this context, this means that the trigger is either triggered or not triggered.

Each listener can have a list of several triggers.

You could say that the whole process consists of two main parts. The first part is about the listeners receiving data, and the second part is about the script engine activating the triggers.

The reception of data looks slightly different depending on whether you have specified a *SOT (Start Of Text)* and/or *EOT(End Of Text)* or not.

Let's start by assuming that neither *SOT* nor *EOT* has been specified. As soon as the listener has received data, it will check its triggers' *Match* string for matching strings in the received data. If no match is found, the listener keeps receiving data that is kept in the listener's internal buffer.

Each time new data is received by the listener, it checks its triggers in the order the triggers are defined in the list. When a match is found, all data received so far is copied to a variable named *TriggerString*. The variable is automatically created for each listener and is available via script. The trigger will be put in a queue that the script engine will process. Several triggers can, in this way, match at the same time for a listener. In that case, they will be put in the queue in the order they are defined in the list. The script engine will process them one at a time. If any of the triggers match, the internal buffer of data will be cleared, and the whole process will start all over again.

When there are triggers in the queue, the script engine will process them one at a time. The trigger first in line will be active during one round of script execution. That means that the corresponding variable is *true* during one execution interval. After that, the variable is set to *false* again, and the trigger is removed from the queue. The corresponding *TriggerString* will also be cleared. Then the script engine takes the next trigger in line from the queue, sets its value to *true* during one round of execution, and so on.

Now, let's assume we have entered a text for *SOT*. Now the listener will discard all data received until the text specified for *SOT* appears in the received data. Then the listener starts to collect received data in its internal buffer and check its triggers in the same way as described above. When we find a match the corresponding trigger is put in the queue for the script engine, the *TriggerString* variable will be updated with the data that has been received so far starting with the *SOT*, and after that, the internal buffer will be cleared, and the listener will start looking for a new *SOT* in the incoming data.

If we specify an *EOT*, the check of triggers will not be performed until the text specified as *EOT* appears in the received data. If there is no match, all data, including the *EOT* received so far, will be discarded, and the process starts all over again. If there is a match, the matching trigger will be put in the queue, the *TriggerString* variable will be updated, and the internal buffer up to and including *EOT* will be cleared. Note that there may be several *EOT* in the received data. In that case, they will be processed one at a time. First, the part of the data up to and including the first *EOT* will be processed. Triggers will be checked and possibly put in the queue. Then that part of the data will be discarded, and the next part of the data up to the second *EOT* will be processed, and so on.

Finally, if both *SOT* and *EOT* are specified, the listener will start buffering data after the *SOT* is received, and then the listener collects data until an *EOT* appears. Not before that, the list of triggers will be checked. In all other aspects, it works in the same way as described above.

In the following pages, we describe how to use these listeners in Ethiris. We assume that you are familiar with the listeners in Admin. For more information on how to create and manage listeners, see the *Admin - Configuration for Ethiris* manual.

## 5.2 Types of listeners

### 5.2.1 TCP-inbound

For an example of the listeners, see the *Admin - Configuration for Ethiris* manual.

## 5.2.2 TCP-outbound

This listener works in the way that Ethiris Server takes the initiative to communicate with the external unit. Ethiris then expects the external unit to start sending data spontaneously.

To make this work, the external unit needs to listen to a specific port, and any firewalls in the external unit must allow incoming connections on the specified port.

In the example, we use a UDP-camera that will send notifications to Ethiris Server as it detects motion.

### Configure the camera

The first thing to do is to configure the camera to listen to a port.

Navigate to the camera's web interface, log on, and click on *Setup*.

#### IPE3500M

The screenshot shows the 'Publisher - Motion Detection' configuration page. On the left is a navigation menu with 'Event Configuration' expanded to 'Motion Detection'. The main content area has several sections: 'Subscriber - DO' with 'All Zone' and 'DO #1' checkboxes; 'Subscriber - Email' with 'Post notification message' and 'Attach a snapshot' checkboxes; 'Subscriber - Multicast' with 'Post notification message' checkbox; 'Subscriber - TCP' which is circled in red and has 'Post notification message' checkbox; 'Subscriber - HTTP' with four checkboxes for HTTP Server #1 through #4; and 'Subscriber - FTP' with 'Post notification message' checkbox. At the bottom are 'Apply' and 'Reset' buttons. The version number 'v1.09.12' is in the bottom right corner.

Figure 5.2 Web interface in UDP-camera

Navigate to *Event Configuration* and then *Motion Detection* in the menus of the camera. Click on *TCP* to configure the camera to send *TCP Events*. You should see the following.

#### TCP Setting

The screenshot shows the 'Configuration' dialog box for TCP settings. The 'Enable TCP Event' checkbox is checked and circled in red. Below it, the 'Name' field contains 'Tcp Event' and the 'Listen Port' field contains '1234', both circled in red. At the bottom are 'OK' and 'Cancel' buttons, with the 'OK' button also circled in red. The version number 'v1.07.03' is in the bottom right corner.

Figure 5.3 Setting for TCP Event in the camera

Check *Enable TCP Event* and set the port on which the camera should listen on. In this case, the camera should listen to port 1234.

Click OK and then check *Post notification message* under *Subscriber - TCP*. Finally, click *Apply* to save the changes in the camera.

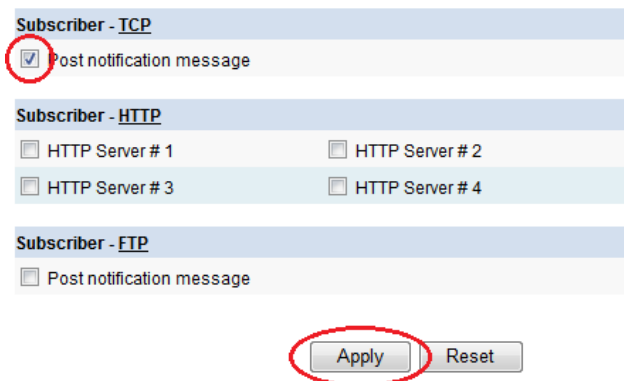


Figure 5.4 Configuration of notification

Now it's time to configure the motion detection in the camera.

Navigate to *Motion Detect* and then to *Zones and Rules* in the cameras web interface.

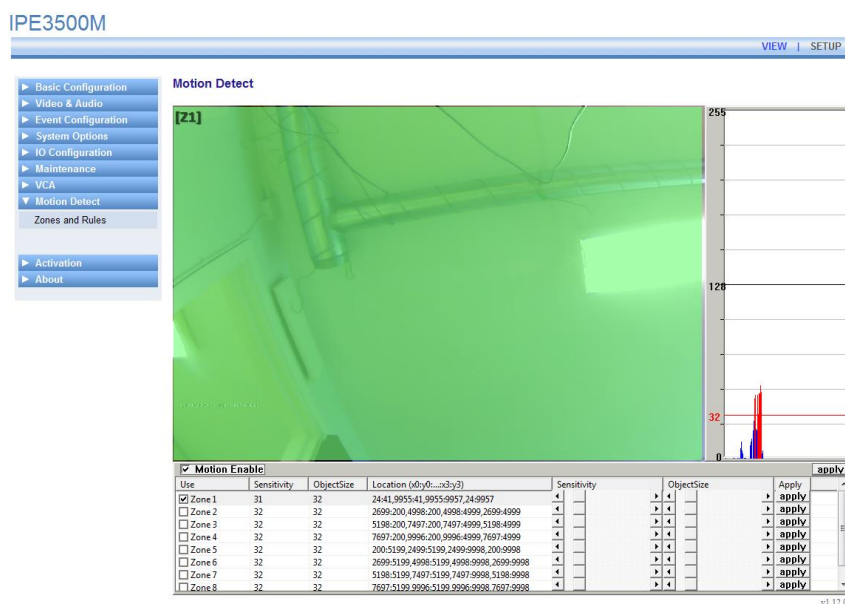


Figure 5.5 Configuration of Motion Detection in the camera

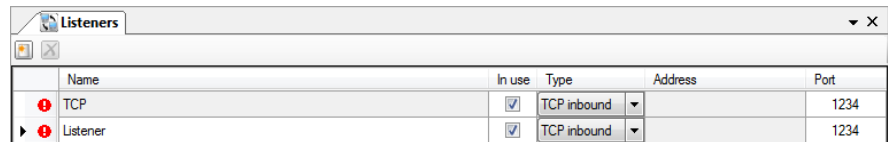
In this case, we only use one zone (Z1) in the camera and it covers the whole image from the camera (the green surface). In the diagram to the right, you can see the amount of motion for the past time. Red means the amount of motion has been above set triggering level. The camera will, in that case, send a notification to the Ethiris Server.

The configuration in the camera is now finished, and we move to configure the listener in Ethiris Admin.

### Configure listener in Ethiris Admin

Double click the node *Listeners* under *Communication* in the treeview.

Then click on *New listener*. You should see something similar to Figure 5.6.

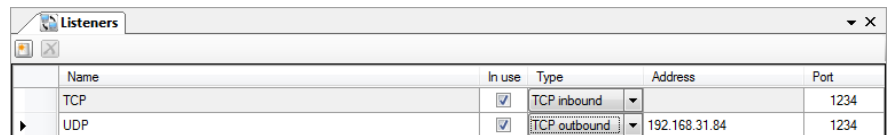


Name	In use	Type	Address	Port
TCP	<input checked="" type="checkbox"/>	TCP inbound		1234
Listener	<input checked="" type="checkbox"/>	TCP inbound		1234

Figure 5.6 The panel Listeners.

Change the name of the listener to UDP and the type to *TCP outbound* as it is Ethisis Server that should take the initiative to the communication. When the type is changed to *TCP outbound*, you also get the possibility to set *Address* to the external unit. In this case, we set the IP-address to the UDP-camera, 192.168.31.84. *Port* shall remain at 1234 as this was the port we configured the camera to listen to, see Figure 5.3.

When you are finished, it looks something like this.

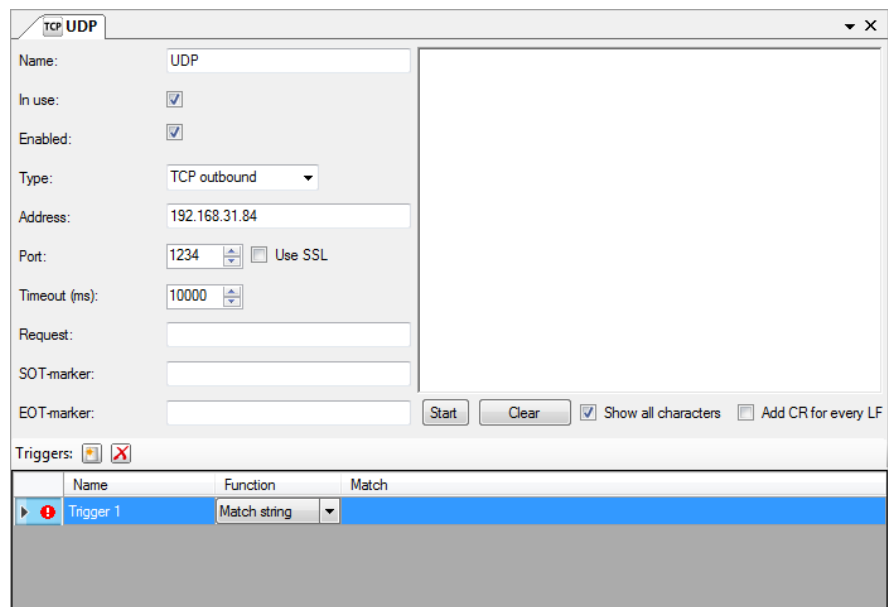


Name	In use	Type	Address	Port
TCP	<input checked="" type="checkbox"/>	TCP inbound		1234
UDP	<input checked="" type="checkbox"/>	TCP outbound	192.168.31.84	1234

Figure 5.7 The panel Listeners with type changed for the new listener.

Note the error mark that is shown in Figure 5.6. This is because two listeners are configured as an *inbound* connection on the same port. This will not work. When the type for the new listener is changed to *outbound*, the error mark disappears.

Now double click on the new listener in the treeview to the left. The listener's panel will be shown.



Name: UDP  
 In use:   
 Enabled:   
 Type: TCP outbound  
 Address: 192.168.31.84  
 Port: 1234  Use SSL  
 Timeout (ms): 10000  
 Request:   
 SOT-marker:   
 EOT-marker:  Start Clear  Show all characters  Add CR for every LF

Name	Function	Match
Trigger 1	Match string	

Figure 5.8 The UDP panel.

Now it's time to do the final configuration of the listener. You can get some help by starting the monitoring of the traffic between the camera and the Ethisis Server. In the traffic, the keywords that we need to identify for the listener to work optimally. Examine the text flow from the camera carefully and look for the keywords we will use as SOT, EOT, to start recording and to stop recording.

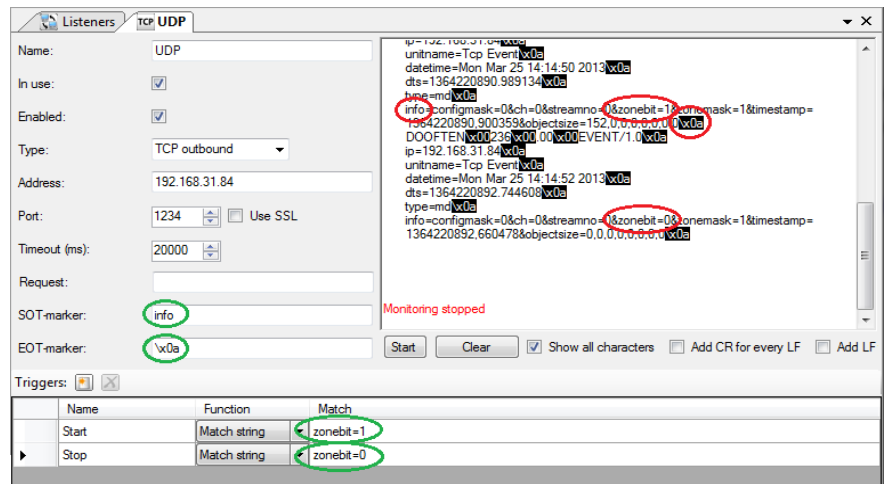


Figure 5.9 The UDP panel with identified keywords.

In the text flow from the camera, the keywords have been identified and marked by red circles. The keywords are then added to the corresponding fields in the listener’s configuration.

The text ‘info’ start each row text from the camera where the keywords for start respective stop of recording are. Hence we set ‘info’ as SOT for the listener.

Likewise, we identify the character ‘\x0a’ as the last character in each row where the keywords for start respective stop of recording are. So, we set ‘\x0a’ as EOT for the listener.

A small detail that can be very important is the setting for *Timeout (ms)*. The camera automatically sends a heartbeat notification approximately every 15<sup>th</sup> second. This is to ensure that the communication channel is up and running. To make sure Ethiris doesn’t alert on communication error with the camera, we must set the timeout at 20s. It works in that way that if Ethiris doesn’t get any data from this listener within set timeout, Ethiris will activate the alarm *Communication error with listener*.

Now it is time to identify the keywords for the triggers we will use to start and stop recording in Ethiris.

In our case, we have only one zone defined in the camera, why we only need to use the text ‘zonebit=1’ as a trigger for motion start. Likewise, we use ‘zonebit=0’ as a trigger for motion end in the camera.

We create two triggers with these keywords to match.

**Script in Ethiris Admin**

The last step is to write that line of script code needed to connect the listener with the camera and to start/stop recording.

The camera we will record from is called UDPCam in our example.

The two triggers show up as variables of type *Boolean* in the variable browser in script panel.

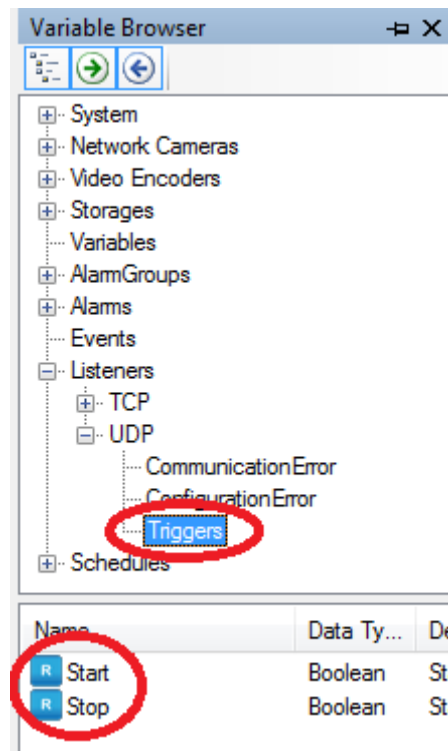


Figure 5.10 The new trigger variables are available in the variable browser.

With these new variables, we can write a small script for starting a recording when there is movement in front of the camera.

The following script makes this happen:

```
UDPCam.RecordEvent = (UDPCam.RecordEvent || UDP.Start)
&& !UDP.Stop;
```

Since the trigger variables are *true* only during one round of script execution, we need this construct of the script where the *RecordEvent* variable holds itself until we get a stop signal in the form of *UDP.Stop* becoming *true*.

|| means "or".

! means "not".



# 6 Explanation of Terms

## ActiveX

ActiveX is a framework for defining reusable software components in a programming language-independent way. Software applications can then be composed of one or more of these components to provide their functionality.

## COM

Component Object Model. A binary communication standard from Microsoft that can be used in communication between various program components. COM is the basis of OPC.

## IP address

Each unit in a computer network has a unique IP address that consists of 4 groups of digits. Each group can be 0-255, for example, 192.168.30.29.

## License key

A key that is received from Kentima after payment has been made, and a registration key has been submitted. The license key is used to “unlock” Ethisis via the license dialog.

## Network camera

A type of camera that can be directly connected to a network. The camera device has a unique IP address.

## OPC

OLE for Process Control. A communication standard developed within the automation industry for communication between different systems.

## Product code

A key following each Ethisis license. The key shall be entered at installation time. The Ethisis server requires a product code at installation.

## Registration key

The key that uniquely identifies an instance of a program. The key is generated by clicking the *Register* button in the license dialog. This key must be sent to Kentima AB to unlock Ethisis.

## TCP/IP

Transmission Control Protocol/Internet Protocol.



Scalability 1:2

# 7 Index

## A

ActiveX 2:1

## C

Configuration

Modbus OPC Server 4:1

## I

Installation

ActiveX 2:2

Modbus OPC Server 4:1

Remote control 3:1

Interface

ActiveX 2:7

Remote control 3:9

## L

Listeners 5:1

Licensing 5:1

TCP-inbound 5:3

TCP-outbound 5:4

Types 5:2

## M

Modbus OPC Server 4:1

## P

Purpose 1:1

## R

Remote control 3:1

## S

Sample

ActiveX 2:3

Remote control 3:2

# KENTIMA PRODUCT LINES

## SECURITY

VIDEO MANAGEMENT SOFTWARE

NETWORK VIDEO RECORDER

PSIM SOFTWARE

## AUTOMATION

HMI/SCADA SOFTWARE

INDUSTRIAL COMPUTERS

OPERATOR PANELS

**Mailing address:**

PO BOX 174  
SE-245 22 Staffanstorp  
Sweden

**Visiting address:**

Kastanjevägen 2  
245 44 Staffanstorp  
Sweden

**Phone:** +46 (0)46-25 30 40

**Fax:** +46 (0)46-25 03 10

**E-mail:** [info@kentima.com](mailto:info@kentima.com)

[www.kentima.com](http://www.kentima.com)

