



Product:	UDPIOServer
Product version:	V1.6
Document ID:	UM-UDPIOSERVER
Doc revision:	A
Written/Appr.:	KD / RE
Date:	6. Nov. 2008

Industrial Control Design AS



UDPIOServer V1.6

User Manual

The content of this document is confidential information not to be published without the consent of Industrial Control Design AS.

Industrial Control Design AS, www.icd.no, support@icd.no, forum.icd.no

Contents

1. INTRODUCTION.....	3
2. APPLICATION CONFIGURATION.....	4
2.1. About.....	4
2.2. How to add the UDPIOServer component to a CDP Application.....	4
3. UDPIOSERVER CONFIGURATION.....	5
3.1. About.....	5
3.2. Generic component configuration.....	5
3.2.1. Signals.....	5
3.2.2. Parameters.....	5
3.2.3. Alarms.....	5
3.3. IOConfig element.....	5
4. UDP PACKET CONFIGURATION.....	6
4.1. Packet element.....	6
4.1.1. NetworkInterface attribute.....	6
4.2. Signal element.....	7
4.3. Bit element.....	7
5. UDPIOSERVER DISPLAYS.....	8
5.1. Web Interface.....	8
6. APPENDIX.....	9
6.1. UDPIOServer example component file.....	9

1. Introduction

This document describes how the UDPIOServer component works, and how to set it up and use it with the CDP system.

The UDPIOServer is used for sending and/or receiving UDP (User Datagram Protocol) packets containing Signal data.

The UDPIOServer component has the following features:

- Input or output UDP packets are defined by packet descriptions in the component XML file. Signals are created for each item in the packet and routing is done like for other CDP signals.
 - **NOTE:** Routing may be set in the UDPIOServer component XML file too, unlike most other IOServers.
- The data in the packets consists of either standard CDP Signals, or registers where each bit may be mapped to either a bool Signal or a component state.
- Send frequency, target address/port, receive port and network interface to use is specified in the XML file.
- Incoming packets may be verified. The default implementation only checks that the packet size is correct. (For more fine-grained verification of packets, please refer to the Programmer Manual)

2. Application Configuration

2.1. About

This chapter describes how to instantiate the UDPIOServer component within a CDP application. For instructions on compiling and linking UDPIOServerLib into a CDP application project, please refer to the Programmer Manual.

2.2. How to add the UDPIOServer component to a CDP Application

Add the following to your project's Application.xml:

Inside the **<Components>** element, add an instance of an UDPIOServer component, for instance:

```
<Component Name="UDPIOServer" src="Components/UDPIOServer.xml"></Component>
```

Or, inside the **<Subcomponents>** element, add:

```
<Subcomponent Name="UDPIOServer" Model="UDPIOServer" src="Components/UDPIOServer.xml"></Subcomponent>
```

This will tell CDP to initialize a component named "UDPIOServer" from a component file located at "Components/UDPIOServer.xml". Make sure that your Models\ folder contains an UDPIOServer.xml model file, or the component will not be initialized correctly. It should also contain the SignalTwoWay_*.xml model files.

3. UDPIOServer Configuration

3.1. About

Configuration is done by modifying the component XML file. It should not be necessary to modify the model XML file(s). See the Appendix for an example UDPIOServer component XML file.

3.2. Generic component configuration

The *send frequency* is specified by the component's standard **fs** element. This means sending is done from the Online state.

The UDPIOServer may be activated and suspended using standard *Activate* and *Suspend* messages.

Unlike most other IOServers, it *is* possible to set the signal routing in the UDPIOServer component XML file.

The UDPIOServer component has these generic component elements:

3.2.1. Signals

Signal	Description
Time	The relative time.

3.2.2. Parameters

Parameter	Description
signalTimeout	Timeout duration before Transmission Error alarm is triggered. (Inherited from IOServer)

3.2.3. Alarms

Alarm	Description
Transmission Error	Alarm raised if we didn't receive packets in time. (Inherited from IOServer)

3.3. IOConfig element

The IOConfig XML element goes inside the Component element and contains all the Packet configurations, both for out-going and incoming UDP packets.

4. UDP Packet Configuration

This chapter describes how to configure the specific UDP packets.

4.1. Packet element

The Packet XML element defines a UDP packet to be sent or received.

The following attributes are special for the **Packet** element:

Attribute	Description
RemoteAddress	IP-address to which the packet will be sent. When this attribute is specified, the packet is considered an output packet, otherwise it is an input packet. A broadcast address may be specified (for instance 10.0.2.255). (See note below).
RemotePort	The port to which the packet will be sent. If not specified, port 2000 is default.
LocalPort	May be specified for input packets to override the default port 2000.
NetworkInterface	Local ethernet interface to use. Must match an interface listed in Application.xml. (See note below)

Example XML:

```
<Packet Name="Outgoing" RemoteAddress="10.0.2.255" RemotePort="2000" NetworkInterface="ETH0">
  <Signal Type="double" Name="Generator" Routing="No" NetworkConvert="1"></Signal>
</Packet>
<Packet Name="Incoming" LocalPort="2000" NetworkInterface="ETH0">
  <Signal Type="short" Name="Joystick" NetworkConvert="1"></Signal>
</Packet>
```

4.1.1. NetworkInterface attribute

Note that the ETH0 interface is *handled specially* in Windows, meaning that it will use the default network interface selected by CDPNetworkSetup.

Also note that listening on 127.0.0.1 is not the same as listening on the computer's IP-address. To be able to listen to 127.0.0.1, create an additional network interface in Application.xml with this IP-address, like this:

```
<!-- NetworkInterfaces -->
<NetworkInterface Name="ETH0" IPAddress="127.0.0.1" SubnetMask="255.255.255.0"></NetworkInterface>
<NetworkInterface Name="ETH1" IPAddress="127.0.0.1" SubnetMask="255.255.255.0"></NetworkInterface>
```

This may look strange, but is necessary since under windows, ETH0 will be changed from 127.0.0.1 to the IP-address chosen by CDPNetworkSetup.

4.2. Signal element

One or more Signal elements go inside the Packet element to make up the data that is transmitted or received using UDP.

The following attributes are special for the **Signal** element in this context:

Attribute	Description
NetworkConvert	May be set to indicate that the data should be converted to/from network format (big-endian format). The default is to not convert.

Example XML:

```
<Signal Type="double" Name="Generator" Routing="Sinus.Output" NetworkConvert="1"></Signal>
```

4.3. Bit element

The **Bit** element is used to specify *registers*, or collection of bits. It goes inside the **Signal** element, where Bit elements define bool signals for each bit. (A bit behaves like a Signal of type bool).

Component states may also be mapped to bits (see example below). The bit will be set to true when the specified component is in the specified state. Remote components can be listed.

Example XML:

```
<Signal Type="unsigned short" Name="Status" NetworkConvert="1">
  <Bit Nr="0" Name="HPU Running" Component="HPU" State="Running"></Bit>
  <Bit Nr="1" Name="Pump1 Running mode" Component="HPU.Pump1" State="Running"></Bit>
  <Bit Nr="2" Name="Pump2 Running mode" Component="HPU.Pump2" State="Running"></Bit>
  <Bit Nr="3" Name="Pump3 Running mode" Component="HPU.Pump3" State="Running"></Bit>
</Signal>
```

5. UDPIOServer Displays

The UDPIOServer status and configuration can be viewed on the Web Interface.

5.1. Web Interface

The web interface lists the packets defined. For each signal, the signal value and byte offset inside the packet is listed.

For bits inside a register, the bit number is listed using the notation byteNr:bitNr.

Note:

The Application\WebServer directory must be from CDP 2.2.0.4 or newer to display the packets as shown in the figure.

Offset:Bit	Signal name	Value
0	:Generator	:0
8	:CableSpeed	:0
16	:CableTension	:0
24	:Alarms	:4
24:0	:Spooling device out of pos	:0
24:1	:SpoolingDevice encoder error	:0
24:2	:SpoolingDevice encoder1 alarm	:1
24:3	:SpoolingDevice encoder2 alarm	:0
25	:Alarms Pumps	:0
25:0	:Pump1 overload	:0
25:1	:Pump1 thermistor	:0
25:2	:Pump2 overload	:0
25:3	:Pump2 thermistor	:0
25:4	:Pump3 overload	:0
25:5	:Pump3 thermistor	:0
26	:Status	:0
27:0	:HPU Running	:0
27:1	:Pump1 Running mode	:0
27:2	:Pump2 Running mode	:0
27:3	:Pump3 Running mode	:0

6. Appendix

6.1. UDPIOServer example component file

```

<?xml version="1.0" encoding="iso-8859-1"?>
<Component Name="UDPIOServer" Model="UDPIOServer">
  <Activate>1</Activate>
  <fs>10</fs>

  <Description>
    <![CDATA[Send and receive preconfigured UDP packets.]]>
  </Description>

  <IOConfig>
    <Packet Name="UDP Out" RemoteAddress="10.0.2.255" RemotePort="2000" NetworkInterface="ETH0">
      <Signal Type="double" Name="Generator" Unit="m/s" Routing="Sinus.Output" NetworkConvert="1"></Signal>
      <Signal Type="double" Name="CableSpeed" Unit="m/s" Routing="Encoder.Output" NetworkConvert="1"></Signal>
      <Signal Type="double" Name="CableTension" Unit="t" Routing="Loadcell.Load" NetworkConvert="1"></Signal>

      <Signal Type="unsigned char" Name="Alarms" NetworkConvert="1">
        <Bit Nr="0" Name="Spooling device out of pos" Routing="No"></Bit>
        <Bit Nr="1" Name="SpoolingDevice encoder error" Routing="No"></Bit>
        <Bit Nr="2" Name="SpoolingDevice encoder1 alarm" Routing="No"></Bit>
        <Bit Nr="3" Name="SpoolingDevice encoder2 alarm" Routing="No"></Bit>
      </Signal>

      <!-- Alarm's SignalOut signal is routed to these signals. -->
      <Signal Type="unsigned char" Name="Alarms Pumps" NetworkConvert="1">
        <Bit Nr="0" Name="Pump1 overload" Routing="No" Description="Pump1 reported overload alarm."></Bit>
        <Bit Nr="1" Name="Pump1 thermistor" Routing="No" Description="Pump1 reported thermistor alarm."></Bit>
        <Bit Nr="2" Name="Pump2 overload" Routing="No" Description="Pump2 reported overload alarm."></Bit>
        <Bit Nr="3" Name="Pump2 thermistor" Routing="No" Description="Pump2 reported thermistor alarm."></Bit>
        <Bit Nr="4" Name="Pump3 overload" Routing="No" Description="Pump3 reported overload alarm."></Bit>
        <Bit Nr="5" Name="Pump3 thermistor" Routing="No" Description="Pump3 reported thermistor alarm."></Bit>
      </Signal>

      <!-- Map component states to bits in a "register" -->
      <Signal Type="unsigned short" Name="Status" NetworkConvert="1">
        <Bit Nr="0" Name="HPU Running" Component="HPU" State="Running"></Bit>
        <Bit Nr="1" Name="Pump1 Running mode" Component="HPU.Pump1" State="Running"></Bit>
        <Bit Nr="2" Name="Pump2 Running mode" Component="HPU.Pump2" State="Running"></Bit>
        <Bit Nr="3" Name="Pump3 Running mode" Component="HPU.Pump3" State="Running"></Bit>
      </Signal>
    </Packet>

    <Packet Name="PacketIn" LocalPort="2000" NetworkInterface="ETH0">
      <Signal Type="unsigned char" Name="BinaryInSignals" NetworkConvert="1">
        <Bit Nr="0" Name="Joystick_Request" Description="Request thruster for command"></Bit>
      </Signal>
      <Signal Type="short" Name="Joystick_Alive" Unit="%" Description="" NetworkConvert="1"></Signal>
      <Signal Type="short" Name="Command_Position_Order" Unit="%" Description="" NetworkConvert="1"></Signal>
    </Packet>

  </IOConfig>

  <Signals>
    <Signal Name="Time" Unit="s" Type="double" Description="The relative time."></Signal>
  </Signals>

  <Alarms>
    <Alarm Name="Transmission Error" Level="warning" Enabled="1" Text="Didn't receive packets in time.."></Alarm>
  </Alarms>

  <Parameters>
    <Parma Name="SignalTimeout" Value="1.1" Unit="s" Description="Timeout before Transmission Error alarm."></Parma>
  </Parameters>
</Component>

```