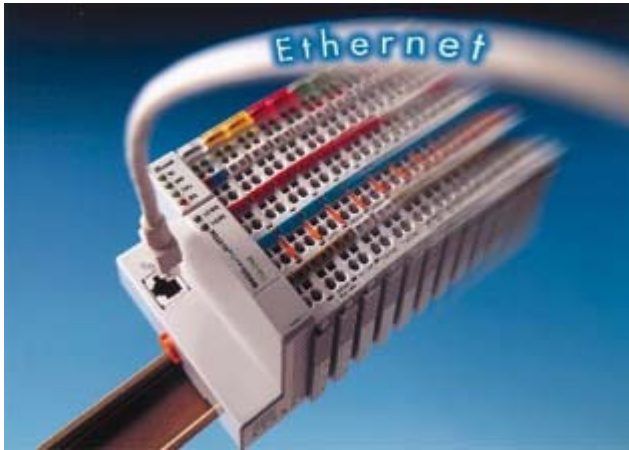




Product:	Wago 750-342 Modbus
Product version:	V1.1
Document ID:	UM-Wago 750-342 Modbus
Doc revision:	A
Written/Apr.:	RE / SL
Date:	10/03/2008

## Industrial Control Design AS



# Wago 750-342 Modbus 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The content of this document is confidential information not to be published without the consent of Industrial Control Design AS.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](http://www.icd.no), [support@icd.no](mailto:support@icd.no), [forum.icd.no](http://forum.icd.no)

# Contents

<b>1. INTRODUCTION.....</b>	<b>3</b>
1.1. About.....	3
<b>2. CONFIGURATION.....</b>	<b>4</b>
2.1. Create Wago IOServer.....	4
2.2. Setting up the XML.....	4
2.3. fs.....	5
2.4. IOConfig/Node.....	5
2.4.1. Description.....	5
2.4.2. Example XML.....	5
2.5. Inputs.....	5
2.5.1. Description.....	5
2.5.2. Example XML.....	5
2.5.3. Elements.....	5
2.6. Outputs.....	6
2.6.1. Description.....	6
2.6.2. Example XML.....	6
2.6.3. Elements.....	6
2.7. AnalogChannel Scaling.....	7
2.8. Alarms.....	7
2.9. Signals.....	7
2.10. Parameters.....	8
2.11. Autoconfiguration.....	8
2.12. Routing.....	8
2.13. Modify the project .xml files.....	8
2.14. Setting up WAGO InitPackets.....	9
2.15. Using the WAGO 750-650 RS232C Module.....	9
<b>3. APPENDIX.....</b>	<b>11</b>
3.1. Example WagoIO XML-file.....	11

# 1. Introduction

## 1.1. About

This document describes how to set up the XML for a Wago 750-342 I/O Module. For more information about Wago devices, please visit <http://www.wago.com/>.

This document requires some knowledge about Modbus ( <http://www.modbus.org/> ), Packets and FunctionDefinitions. Please see the CDP documentation 'UM-Modbus Setup' and 'UM-Packets and FunctionDefinitions ' for more information.

The Wago 750-342 is an Input/Output module that can communicate using Modbus request/reply packets. It is connected via TCP/IP, UDP/IP or serial RS232 and can be operated at speeds up to approximately 200 Hz.

## 2. Configuration

Configuration is done by modifying the component xml file inside the Application\Components\ folder. It should not be necessary to modify the model xml file. An example of WagoIO.xml file (component xml) is found in 3.1.

### 2.1. Create Wago IOserver

To communicate with the Wago 750-342, you must instantiate a Modbus IO-server, by adding the following to Application.xml:

After the <Console> tag:

```
<Definitions>
  <Definition src="Models\ModbusFunctions.xml"></Definition>
</Definitions>
```

This will tell CDP to load the function-definitions for modbus communication, and some special functions for the WAGO I/O Module (watchdog settings).

Inside the <Subcomponents> tag, add an instance of a ModbusTCPIOServer:

```
<Subcomponent Name="WagoIO" Type="ModbusTCPIOServer"></Subcomponent>
```

This will tell CDP to initialize a component named “WagoIO” of type ModbusTCPIOServer.

If you prefer to use UDP as transfer protocol for data, create a ModbusUDPIOServer instead:

```
<Subcomponent Name="WagoIO" Type="ModbusUDPIOServer"></Subcomponent>
```

CDP will look in a folder with the name of the application for the xml file you specified. If your application is called “Test”, CDP will try to open “Components\Test\WagoIO.xml”. If you want the file to be placed elsewhere and/or you want to name is something else, you can specify a path and name of the file by using the src="" attribute:

```
<Subcomponent Name="WagoIO" Type="ModbusTCPIOServer" src="Components\Wago750-342.xml"></Subcomponent>
```

### 2.2. Setting up the XML

To set up the WAGO module correctly, you have to know which I/O-modules are connected to it, and whether they are analog or digital. You also need to know how many registers (i.e. 16-bit words) each module occupy.

The Wago module is organized like this: Analog and Digital Output addresses start at 0. Analog and Digital Input addresses also start at 0. This means that the first output module is written to at address 0, and the first input-module is read from at address 0.

Address 0 can be seen as a virtual address that is an output-module on write, and an input-module on read.

Please note that the physical order of the I/O modules is important.

Digital Modules always get their addresses after the analog modules if you use the read/write registers functions. This can be seen from the illustrations on page 281 in the 750-342 WAGO manual.

## 2.3. fs

The <fs> element specifies the frequency of which to run the reading and writing of the Input/Output values. <fs>100</fs> means 100 Hz, so a read and write is done every 10 milliseconds. If the runningTooFast alarm is constantly set, then you could have specified a too high fs. Typically, the maximum fs for the Wago IPC is 200 Hz.

## 2.4. IOConfig/Node

### 2.4.1. Description

IOConfig is an XML element that wraps the Input / Output configuration. A minimal configuration is shown below.

### 2.4.2. Example XML

```
<IOConfig>
  <Node Name="WagoIO">
    <Inputs>
    </Inputs>
    <Outputs>
    </Outputs>
  </Node>
</IOConfig>
```

## 2.5. Inputs

### 2.5.1. Description

Contains the ChannelGroups and Channels / signals to receive input from the physical Wago Input Modules.

### 2.5.2. Example XML

```
<Inputs>
  <ChannelGroup Type="Analog" NumberOf="2" ModuleNr="0">
    <Channel Nr="0" Type="short" Name="750-454 2AI 0.0"></Channel>
    <Channel Nr="1" Type="short" Name="750-454 2AI 0.1"></Channel>
  </ChannelGroup>
</Inputs>
```

### 2.5.3. Elements

Element	Description
ChannelGroup	An enclosing element for a group of channels/signals.
Channel	A Signal / Channel that correspond to one value received in.

ChannelGroup	Description
Type	Can be 'Analog' or 'Digital'
NumberOf	The Number of channels in this ChannelGroup
ModuleNr	The physical module number, used only for documentation.
SizeInBytes	Only for Digital ChannelGroups: Specifies the size in bytes of the ChannelGroup. For the Wago IPC this is always set to 2.

Channel Attribute	Description
Number	The number in a sequence, must be last number+1. On Digital channels this also signifies the bit position, starting at 0.
Type	The c++ data type, can be bool, char, byte, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long, float or double. Typical values are bool and short.
Name	The signal name for this channel. Feel free to use more understandable names than what is used in the example, like 'Pressure', 'Oil Level' and so on.

## 2.6. Outputs

### 2.6.1. Description

Contains the ChannelGroups and Channels/signals to send as output from the Wago Modules.

### 2.6.2. Example XML

```

<Outputs>
  <ChannelGroup Type="Analog" NumberOf="2" ModuleNr="0">
    <Channel Nr="0" Type="short" Name="750-454 2AI 0.0"></Channel>
    <Channel Nr="1" Type="short" Name="750-454 2AI 0.1"></Channel>
  </ChannelGroup>
</Outputs>

```

### 2.6.3. Elements

Element	Description
ChannelGroup	An enclosing element for a group of channels/signals.
Channel	A Signal / Channel that correspond to one value received in.

ChannelGroup	Description
Type	Can be 'Analog' or 'Digital'
NumberOf	The Number of channels in this ChannelGroup
ModuleNr	The physical module number
SizeInBytes	Only for Digital ChannelGroups: Specifies the size in bytes of the ChannelGroup. For the Wago IPC this is always set to 2.

Channel Attribute	Description
Number	The number in a sequence, must be last number+1. On Digital channels this also signifies the bit position
Type	The c++ data type, can be bool, char, byte, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long, float or double. Typical values are bool and short.
Name	The signal name for this channel.

## 2.7. AnalogChannel Scaling

Analog Channels can do multipoint scaling each time they are read or written. Please see the document 'AnalogChannel with Multipoint scaling.pdf' in the Doc folder where you installed CDP for more information about this. On a Windows install, there is also a shortcut placed on 'Start Menu'-'>CDP'-'>Doc'-'>IOServers'.

## 2.8. Alarms

The following alarms can trigger from this IOserver:

Alarm Name	Description
Transmission Error	An error is causing the transmission of signals to fail
RunningTooFast	The <fs> is set too high, the I/O can not keep up.
ModuleError	There is an error on one or more of the IO Nodes that causes the process image update to fail

In addition, you can set up alarms to trigger directly on a Channel mask.

A Channel Alarm is set up in XML like this:

```
<Channel Nr="0" Type="short" Name="750-454 2AI 0.0" ErrorMask="0x0003"
AlarmMessageCommand="Stop" AlarmMessageDestination="..ControlCode" AlarmText="Cable break on
Valve feedback" Description="A Cable break was detected on the first input module in the Winch
IO Cabinet (cable from winch speed valve feedback)"></Channel>
```

The Alarm will get the name of the channel, and " Alarm" is appended to that name. For this reason, make sure the Channel Name is less than 25 characters to avoid the problem with ShortName>31 characters.

Channel Alarm Attribute	Description
ErrorMask	Required. Specifies a Mask to AND (&) with the channel. If the result is non-zero, the alarm is set (see Timeout below)
AlarmMessageCommand	MessageCommand to send when alarm is set.
AlarmMessageDestination	MessageDestination for the AlarmMessageCommand.
AlarmText	Text to set in the Alarm, visible in a visualizer
AlarmDescription	Description to set in the alarm, visible in a visualizer
Timeout	Time in seconds that the error condition must be active, before the alarm is set. Default value 0.

## 2.9. Signals

The following signals are in the IOserver:

Signal Name	Description
Send-Receive Roundtrip time	The time used for outputing and inputing data, including signal conversion.
ReadTimeStamp	The globalTime when last read happened
WriteTimeStamp	The globalTime when last write happened
OutputDisabled	Only used in conjunction with CDP Redundancy: Is set to 1 if parameter 'RD output disable control' is set to 1 and the RDmanager is not in the Active state, else it is 0. This signals tells if the outputs are written out to the physical modules(if value=0) or not (value=1).

## 2.10. Parameters

Parameter Name	Description
doReadFirst	Set to 1 to perform reads before writes.
RD output disable control	Only used in conjunction with CDP Redundancy: Set to 1 to make RDManager disable output if RDManager is not in Active State.
AutoConfig (see below)	Set to 1 to Automatically adjust the .xml file to reflect the actual layout on a change in configuration, or when the controller boots up. Please note that all the xml for physical modules is regenerated when AutoConfig is set to 1, causing you to loose existing settings.

## 2.11. Autoconfiguration

AutoConfiguration can be enabled by setting parameter AutoConfig to 1. When adding a new Module to the Wago IPC, or when removing one, the XML is then updated to reflect the change. This is the preferred method of setting up the WagoIO the first time. After configuring, you can set AutoConfig="0" and then modify the names of the channels to suit your needs.

## 2.12. Routing

To get the IOServer to output sensible signals, you will have to route the signals from another component. In CDP version 2.3.1.0 and earlier you can **not** set routing directly on an IOServer such as this. You will have to 'push' the routing from a signal on the same controller as the IOServer is located. 'Push' routing only works on components running inside the same controller. For instance, a 'Sinus' component could have routing on its 'Output' signal set to 'WagoIO.Analog Output 1', this will effectively 'write' the sinus out to the Module containing that signal.

## 2.13. Modify the project .xml files

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

Inside the <Components> element, add an instance of a WagoIO component, for instance:

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

This will make the component a so-called 'top-level' component, quickly accessible from anywhere in the network. In a CDPBrowser, the component will then be visible on the same level as other Application components.

If you want to have the component 'hidden' inside your application, add the following inside the <Subcomponents> element:

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

This will tell CDP to initialize a component named "WagoIO" from a component file located at "Components/WagoIO.xml", and place it below the Application component.

Make sure that your Models\ folder contains a WagoIO.xml model file, or the component will not be initialized correctly.

The WagoIO Model file can be found in \$(CDPBase)\Templates\Project\_template\Application\Models\, and an example WagoIO component file can be found in \$(CDPBase)\Templates\XML Templates\IO Servers.

## 2.14. Setting up WAGO InitPackets

The WAGO watchdog is set up by specifying the following inside the <Node> tag of the Modbus IO-server component .xml file:

```

<!-- Packets to send to I/O after a successful connect (i.e. set up watchdog...) -->
<InitPackets>
  <Packet Name="StopWAGOWD1" FunctionCode="MODBUSStopWAGOWatchDogStep1"
NetworkConvert="1"></Packet>
  <Packet Name="StopWAGOWD2" FunctionCode="MODBUSStopWAGOWatchDogStep2"
NetworkConvert="1"></Packet>
  <Packet Name="SetWAGOTimeOut" FunctionCode="MODBUSSetWAGOTimeout500ms"
NetworkConvert="1"></Packet>
  <Packet Name="StartWAGOWatchDog" FunctionCode="MODBUSStartWAGOWatchDog"
NetworkConvert="1"></Packet>
</InitPackets>

```

The <InitPackets> element is used to tell the IOserver a sequence of packets to send after a successful connect to the IOserver. These will be sent each time CDP loses connection to the IOserver. As you can see from the packets above, they contain no inputs nor outputs. This is because they contain no dynamic data, and are described completely in the <Models\ModbusFunctions.xml> file. It is important that you remember to add the ModbusFunctions.xml to the definition-manager, as explained in Chapter 1.2 Create WagoIOserver.

## 2.15. Using the WAGO 750-650 RS232C Module

CDP has support for receiving and sending *strings* from/to the Wago 750-650 module on serial RS-232C interface. Any number of modules can be used for input of data, but note that output strings (received as CM\_TEXTSRING into the IOserver component) will be sent to *all* 750-650 modules on an IO unit. The serial module is very special, as it is actually an analog input and output module, with 4 bytes input and 4 bytes output. Special codes and data are put into the output register to read/write data, while the input register is used to read data. This is all handled by the ModbusTCPIOServer / ModbusUDPIOServer.

The IOserver needs to uniquely identify this module. This is done by specifying the Channel tags:

- **ModuleType**=”WAGO 750-650”: Must be specified exactly like this to identify the module correctly.
- **InstanceNr**=”<number>”: This number is used to match the input and output part of the module. InstanceNr must match in both the input and output elements of a packet, and must be unique for the module on the IOserver. If you have several WAGO 750-650 modules in an IOserver, give each new module a unique number, but make sure that the input part of the module has an InstanceNr that matches the output InstanceNr of the module.
- The channel **Type**=”unsigned int” because the input/output (analog) size of the module is 4 bytes each.

Inside each channel, a string element must be specified. The **String** element has these tags:

**Header:** Specifies the start of the string. Any string(s) that match this header will be further analyzed to see if the Footer also match.

**Footer:** the end of the string. If both header and footer match, and the size is less than MaxLength, the string is sent to the object specified in MessageDestination.

**MaxLength:** If the string length exceeds this length, it is discarded.

**MessageDestination:** Name of a CDPCComponent or CDPObject that will receive a CM\_TEXTSTRING containing the string.

Note that, in the Header and Footer, you can specify “\*” to accept any character, and you can specify “{hexvalue}” to insert the hex number of a character. For instance, to specify the character “ (hex value 0x22), put “{0x22}” where you want it.

The IOserver receives CM\_TEXTSTRING messages which will be sent to all the RS232 modules on the IOserver. The incoming textstrings are queued up in the IOserver and sent one after another.

The following xml shows how to set up the WAGO 750-650 module in a packet:

```
<Packet Name="IOsignalsUDP" FunctionCode="MODBUSReadWriteMultipleRegisters" NetworkConvert="1">
  <Inputs>
    <ChannelGroup Type="Analog" NumberOf="1" Offset="0" ModuleNr="0">
      <Channel Nr="0" Type="unsigned int" Name="750-650 IN" ModuleType="WAGO 750-650"
        InstanceNr="0">
        <String Header="$**TEST" Footer="{0x0d}" MaxLength="82"
MessageDestination="TestController.SerialStringDispatcher"></String>
        <String Header="This must match exactly" Footer="{0x22}" MaxLength="82"
MessageDestination="TestController.WagoIO"></String>
      </Channel>
    </ChannelGroup>
  </Inputs>
  <Outputs>
    <ChannelGroup Type="Analog" NumberOf="1" Offset="0" ModuleNr="0">
      <Channel Nr="0" Type="unsigned int" Name="750-650 OUT" ModuleType="WAGO 750-650" InstanceNr="0">
      </Channel>
    </ChannelGroup>
  </Outputs>
</Packet>
```

Please note that the 750-650 module is restricted by the communication-speed, as it can only send/receive 3 characters per fs. This means that if you communicate at 10 Hz, max 30 characters per second can be sent/received.

# 3. Appendix

## 3.1. Example WagoIO XML-file

```

<?xml version="1.0" encoding="utf-8"?>
<Component Name="WagoIO" Type="ModbusTCPIODevice">
  <Activate>1</Activate>
  <IOConfig>
    <Node Name="Wago 750-342">
      <Type>ModbusTCP</Type>
      <Number>1</Number>
      <NetworkInterface LocalName="ETH0" RemoteIP="10.0.2.12" SubnetMask="255.255.255.0" RemotePort="502"></NetworkInterface>
      <fs>100</fs>
      <!-- Packets to send to I/O after a successful connect (i.e. set up watchdog...) -->
      <InitPackets>
        <Packet Name="StopWAGOWD1" FunctionCode="MODBUSStopWAGOWatchDogStep1" NetworkConvert="1"></Packet>
        <Packet Name="StopWAGOWD2" FunctionCode="MODBUSStopWAGOWatchDogStep2" NetworkConvert="1"></Packet>
        <Packet Name="SetWAGOTimeOut" FunctionCode="MODBUSSetWAGOTimeout500ms" NetworkConvert="1"></Packet>
        <Packet Name="StartWAGOWatchDog" FunctionCode="MODBUSStartWAGOWatchDog" NetworkConvert="1"></Packet>
      </InitPackets>
      <!-- Packets to send to I/O on modbus error -->
      <ErrorPackets>
        <!-- No need for this on the WAGO I/O Modules -->
      </ErrorPackets>
      <Packet Name="WagoIOSignals" FunctionCode="MODBUSReadWriteMultipleRegisters" NetworkConvert="1">
        <Inputs>
          <ChannelGroup Type="Analog" NumberOf="2" Offset="0" ModuleNr="0">
            <Channel Nr="0" Type="short" Name="AnalogInput1" Min="0" Max="32767">
              <Unit Name="A" HWLow="0" HWHigh="32767" UnitLow="0" UnitHigh="40"></Unit>
            </Channel>
            <Channel Nr="1" Type="short" Name="AnalogInput2" Min="0" Max="32767">
              <Unit Name="V" HWLow="0" HWHigh="32767" UnitLow="0" UnitHigh="100"></Unit>
            </Channel>
          </ChannelGroup>
          <ChannelGroup Type="Digital" NumberOf="2" Offset="2" ModuleNr="2" SizeInBytes="2">
            <Channel Nr="0" Type="bool" Name="DigitalInput1" Invert="false"></Channel>
            <Channel Nr="1" Type="bool" Name="DigitalInput2" Invert="false"></Channel>
          </ChannelGroup>
        </Inputs>
        <Outputs>
          <ChannelGroup Type="Analog" NumberOf="2" Offset="0" ModuleNr="1">
            <Channel Nr="0" Type="unsigned short" Name="AnalogOutput1" Min="0" Max="65535" DefaultValue="33057">
              <Unit Name="Step" HWLow="0" HWHigh="65535" UnitLow="0" UnitHigh="65535"></Unit>
            </Channel>
            <Channel Nr="1" Type="unsigned short" Name="Analogoutput2" Min="0" Max="65535" DefaultValue="33057">
              <Unit Name="Step" HWLow="0" HWHigh="65535" UnitLow="0" UnitHigh="65535"></Unit>
            </Channel>
          </ChannelGroup>
          <ChannelGroup Type="Digital" NumberOf="1" Offset="2" ModuleNr="3" SizeInBytes="2">
            <Channel Nr="0" Type="bool" Name="DigitalOutput1" Invert="false"></Channel>
          </ChannelGroup>
        </Outputs>
      </Packet>
    </Node>
  </IOConfig>
  <Parameters>
    <Parma Name="SignalTimeout" Value="1" DefaultValue="0" PreviousValue="0" TimeLastChanged="Thu Nov 20 16:39:45 2003"
    Description=""></Parma>
  </Parameters>
</Component>

```