



Product:	MSI/CSI client V2.2.1
Document ID:	UM - MSI/CSI client
Document type:	User Manual
Doc revision:	A
Date:	28. Oct. 2008
Pages:	9

Industrial Control Design AS

MSI/CSI client V2.2.1

User Manual

Description of how to set up and use the MSI/CSI client component to communicate with an MSI or CSI server.

Updated for MSILib V2.2.1.

The content of this document is confidential information not to be published without the consent of Industrial Control Design AS.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



Contents

1. INTRODUCTION.....	3
1.1. References.....	3
1.2. Features of the MSI component.....	3
1.2.1. Version 1.....	3
1.2.2. Version 2.0.....	3
1.2.3. Version 2.1.....	3
1.2.4. Version 2.2.....	3
1.2.5. Version 2.2.1.....	3
2. SETUP OF THE MSI CLIENT.....	4
2.1. Copy files to their correct locations.....	4
2.2. Modify your project to use the MSILib.....	4
2.3. Modify the project xml files.....	4
3. ADDITIONAL FEATURES.....	8
3.1. Publishing multiple objects.....	8
3.2. TimeConstrained federate.....	8
3.3. Messaging/interactions.....	9
3.4. Debugging.....	9

1. Introduction

This document describes how the MSI client component (MSI IOServer in CDP terms) works, and how to set it up and use it with the CDP system.

1.1. References

ID	Name	Description
1	http://msi.sourceforge.net/	Description of MSI, by Aron Rubin.
2	csidocumentation.pdf	Common Simulation Interface Documentation and Tutorial (preliminary), by Micolai Husteli/MARINTEK, of 28.10.2005. Distributed with OSC version of CSI server.

1.2. Features of the MSI component

1.2.1. Version 1

- Publishes one object list in the component xml file. For the object, a list of signals is specified, and updates for these signals will be sent to the MSI server periodically.
- Subscribes to objects listed in the xml file. For each object, signals may be specified to subscribe to only a subset of an object's signals, or AutoCreateSignals may be set, allowing the MSI client to create the signals as updates for new signals are received.
- The tone from the MSI server is available as a signal named "Tone" under the MSI component.
- Other CDP components may be synchronized by the MSI server's tone signal, allowing them to run in "time constrained" mode. The components will then be scheduled by the MSI client instead of the usual scheduler (CDPEngine).
- The published and the subscribed signals are created as signals under the MSI client, and may be routed and monitored as other CDP signals.
- Data is received on the specified port. The Header/CommandString is checked and the packet is accepted or rejected.

1.2.2. Version 2.0

- Support for CSI server protocol.
- Publishing of more than one object per client is supported.

1.2.3. Version 2.1

- Support for sending and receiving <message ...> events

1.2.4. Version 2.2

- Support for bool attribute values "true" and "false" (case insensitive on receive).

1.2.5. Version 2.2.1

Code fixes for compatibility with CDP 2.3.0.0 and later.

2. Setup of the MSI client.

This guide assumes that the MSILib is kept as a separate library. An alternative to this is to merge the MSI component into another library of standard components.

2.1. Copy files to their correct locations

Copy the example component.xml file that came with the distribution into your project's Components folder, and the model.xml file into the Models folder.

Copy the MSILib folder (source for the MSI library) either into your project, or to a location for standard libraries.

2.2. Modify your project to use the MSILib

Include the MSILib.h header file in your *libraries.h* file:

```
#include <MSILib.h> // If the path to the MSILib source files is added to project settings
or
#include "../MSILib/MSILib.h" // If the MSILib project is copied into the project folder.
```

If the first method is used, for Visual Studio to find the header file, add the path to where the lib source code was unpacked to "CDP_Application->Properties->C/C++->General->Additional Include Directories".

Link the MSILib.lib into your application either by setting the CDP_Application project dependent of the MSILib project (if using the source code version and the MSILib project is added to the solution), or by putting MSILib in the \$(CDPBase)\Libs folder and specifying MSILib_Release.lib (MSILib_Debug.lib) under "CDP_Application->Properties->Linker->Input->Additional dependencies" (for the CDP_Application project).

2.3. Modify the project xml files

Add the following to Application.xml:

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

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

Or, inside the <Subcomponents> element, add:

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

This will tell CDP to create and initialize a component named "MSI" from a component file located at "Components/MSI.xml". Make sure that your Models\ folder contains an MSI.xml model file, or the component will not be initialized correctly.

The MSI **model** file looks like this:

```

<?xml version="1.0" ?>

    <!-- MSI component model. -->
<Model>
  <Type>MSI</Type>
  <ModelTypeClass>C++</ModelTypeClass>
  <BaseModel>IOServer</BaseModel>
  <Priority>normal</Priority>
  <fs>2</fs>

  <TypeHelp>MSI interface.</TypeHelp>

  <States>
    <State Name="Null" Description="Default state. May or may not be used."></State>
    <State Name="Online" Description="Connection to MSI server established."></State>
    <State Name="Offline" Description="No connection to the MSI server."></State>
  </States>

  <StateTransitions>
    <StateTransition FromState="Null" ToState="Offline" Description="Handles initial state-transition at startup."></StateTransition>
    <StateTransition FromState="Online" ToState="Offline" Description="Connection to the MSI server is broken."></StateTransition>
    <StateTransition FromState="Offline" ToState="Online" Description="Connection to the MSI server is established."></StateTransition>
  </StateTransitions>

  <Messages>
    <Message Name="SendMessage" Description="Send message to MSI objects or types. Parameter: the text string that goes into the message, including name, type and parameters."></Message>
  </Messages>

  <Signals>
    <Signal Name="Send-Receive Roundtrip time" Unit="s" Input="0" Type="double" Description="Period of the send task."></Signal>
    <Signal Name="Tone" Unit="s" Input="0" Type="double" Description="Time sync signal from MSI server."></Signal>
  </Signals>

  <Alarms>
    <Alarm Name="Broken signal routing" Level="Error" Enabled="1" Set="0" Unacknowledged="0" Text="Something is wrong with the routing for one or more of the signals." Description="Something is wrong with the routing for one or more of the signals."></Alarm>
    <Alarm Name="Signal not updated" Level="Error" Enabled="1" Set="0" Trig="1" Unacknowledged="0" Text="The signal is not updated (time stamp too old). Probably lost connection with i/o." Description="The signal is not updated (time stamp too old). Probably lost connection with i/o."></Alarm>
    <Alarm Name="Transmission Error" Level="Warning" Enabled="1" Set="0" Unacknowledged="0" Text="ModbusTCPIOServer transmission-error alarm" Description="Transmission-error alarm"></Alarm>
  </Alarms>

  <Parameters>
    <Parma Name="SignalTimeout" Unit="s" Value="1.00" DefaultValue="1.00" Description="Timeout before entering offline state."></Parma>
    <Parma Name="Use MSI format" Unit="s" Value="0.0" DefaultValue="0.0" Description="Set to 1 to use (old) MSI format when talking to server."></Parma>
  </Parameters>

  <RemoteComponents>
</RemoteComponents>
</Model>

```

There shouldn't be any need to modify this file unless the MSI component is modified.

As can be seen from the model file, the MSI model has three states, Null (startup state), Offline (when no connection to the server has been established), and Online (when communication to the server is established, *and all subscribed signals are being updated periodically*).

The “Tone” signal is set to the tone time value received from the MSI server.

The “Send-Receive Roundtrip time” signal is actual the period of the send task (the name is misleading because the signal is inherited from the base class).

The <fs> and <Priority> set by the model file only affects how often the check for communication status is performed. The frequency of the send task is set by the <SendFrequency> element in the component file (described below). Sending and receiving is performed in separate threads. The priority of both is hardcoded in software to CDP_THREAD_PRIORITY_ABOVE_NORMAL.

The MSI **component** xml file can look like this:

```
<Component Name="MSI" Type="MSI">

  <Activate>1</Activate>

  <IOConfig>

    <ServerIP>127.0.0.1</ServerIP>
    <ServerPort>5678</ServerPort>
    <Digits>8</Digits>          <!-- Number of digits in numbers that are output as strings. -->
    <Precision>-1</Precision>   <!-- -1 means "don't truncate", 3 means 3 digits after the decimal point etc. -->
    <SendFrequency>10</SendFrequency> <!-- Published signals will be updated at this frequency\ -->
    <SlowDownToRealTime>1</SlowDownToRealTime> <!-- Delays sending tone response until the same amount of time has passed -->

    <!-- For Publish section, both Object and Type must be specified.
         If Component is specified, it's periodic method (Process...()) will be scheduled by this MSI component. It is then crucial that the
         specified component's has it's priority set to 'none' (in the model file).
         If TimeConstrained="1", Component must be specified. -->
    <Publish Object="Sinus_Sim" Type="SignalSource">
      <!--
        Name:      Local name used to reference the published signal. A signal with the name objectname.name will be created as a
                   subobject of this MSI component.
        Type:      (Optional). Numeric type of the signal to publish. If not specified, double is used (64 bit).
        Routing:   Signal to publish.
        MSIName:   (Optional). Name to use when talking to the MSI server. If not specified, Name will be used.
        Description: Help string. For documentation. -->
      <Signal Name="Pos" Unit="m" Input="1" Type="double" Routing="Sinus.Output" MSIName="Pos" Description="Text"></Signal>
    </Publish>

    <!-- Specify Object or Type. If both are specified, a subscribe for the Type is requested from the MSI server. -->
    <Subscribe Object="MRU">
      <!--
        Name:      Name of the signal to create locally (as a child object of this MSI component).
        Type:      (Optional). Numeric type of the signal to publish. If not specified, double is used (64 bit).
        Routing:   (Optional. Normally the routing will be set in the component that will read this signal.)
        MSIName:   (Optional). May be specified if the signal to subscribe to has another name than specified in the Name attribute.
        Description: Help string. For documentation. -->
      <Signal Name="Position" Unit="m" Input="0" Type="double" Description="Input from MSI server"></Signal>
      <Signal Name="Speed" Unit="m/s" Input="0" Type="double" Description="Input from MSI server"></Signal>
      <Signal Name="Acceleration" Unit="m/s^2" Input="0" Type="double" Description="Input from MSI server"></Signal>
    </Subscribe>

  </IOConfig>

  <Signals>
    <Signal Name="Send-Receive Roundtrip time" Unit="s" Input="0" Type="double" Description="Period of the send task."></Signal>
    <Signal Name="Tone" Unit="s" Input="0" Type="double" Description="Time sync signal from MSI server."></Signal>
  </Signals>

  <Parameters>
    <Parma Name="SignalTimeout" Unit="s" Value="1.0" DefaultValue="1.0" PreviousValue="1.00" TimeLastChanged="Thu Nov 20
    16:39:45 2003" Description="Timeout before entering offline state."></Parma>
    <Parma Name="Use MSI format" Unit="s" Value="0.0" DefaultValue="0.0" PreviousValue="0.00" TimeLastChanged="Tue Nov 14
    16:39:45 2006" Description="Set to 1 to use (old) MSI format when talking to server."></Parma>
  </Parameters>

  <Alarms>
    <Alarm Name="Broken signal routing" Level="Error" Enabled="1" Set="0" Unacknowledged="0" Text="Something is wrong
    with the routing for one or more of the signals." Description="Something is wrong with the routing for one or more of the
    signals."></Alarm>
    <Alarm Name="Signal not updated" Level="Error" Enabled="1" Set="0" Trig="1" Unacknowledged="0" Text="The signal is not
    updated (time stamp too old). Probably lost connection with i/o." Description="The signal is not updated (time stamp too old). Probably
    lost connection with i/o."></Alarm>
    <Alarm Name="Transmission Error" Level="Warning" Enabled="1" Set="0" Unacknowledged="0" Text="ModbusTCPIOServer
    transmission-error alarm" Description="Transmission-error alarm"></Alarm>
  </Alarms>
</Component>
```

The following settings are special for the MSI, and should reside inside the <IOConfig> element:

ServerIP: IP-address of the MSI server. DNS names may be used if a DNS server is on the network.

ServerPort: The port on the server to which the client will try to make a connection.

Digits: Specifies the maximum number of digits to use when converting the double value to a string. Default value is 12.

Precision: Number of digits after the decimal point when converting the numbers to strings. Set the value to -1 (default) to prevent any truncation (full precision).

SendFrequency: Updates for published signals are sent with this frequency to the MSI server.

SlowDownToRealTime: Set to 1 to make the MSI client delay sending tone command to the server before time has passed. Prevents a simulation from running faster than real time.

Publish: Describes an object to publish. For instance:

```
<Publish Object="Sinus_Sim" Type="SignalSource">
  <Signal Name="Sinus" Unit="m/s" Input="1" Type="double" Routing="Sinus.Output" MSIName="Sin" Description="Text."></Signal>
</Publish>
```

Subscribe: Lists an object that this MSI client should subscribe to from the MSI server. Several <Subscribe> elements may be listed to subscribe to several objects from the MSI server. Set AutoCreateSignals="0" to prevent automatic creation if updates are received for other signals than the one listed inside the <Subscribe> section (AutoCreateSignal has default value "1").

```
<Subscribe Object="MRU" AutoCreateSignals="1">
  <Signal Name="Position" Unit="m" Input="0" Type="double" Description="Input from MSI server"></Signal>
  <Signal Name="Speed" Unit="m/s" Input="0" Type="double" Description="Input from MSI server"></Signal>
  <Signal Name="Acceleration" Unit="m/s^2" Input="0" Type="double" Description="Input from MSI server"></Signal>
</Subscribe>
```

The attributes used inside the <Signal ..> element is the same as for other CDP signals, except that the MSIName attribute may be used if another name than Name should be used while talking to the MSI server. If MSIName is not specified, Name will be used instead.

To subscribe to objects by type, for instance all Thruster objects, do this:

```
<Subscribe Type="Thruster"></Subscribe >
```

This will automatically create all signals for all Thruster objects as updates are received from the MSI server. This is a simple, but effective way to be able to view all signals from all Thruster object through the CDPBrowser; just add the line above to the MSI component xml file, and all the signals will be created under the MSI components automatically.

For each signal listed under a <Publish..> or <Subscribe...> section, a signal will be created as a subobject of the MSI object with the name <ObjectName>.<SignalName>. For the example above, a signal with name "Sinus_Sim.Sinus" will be created as a subobject of the MSI component.

Signals in the <Publish> section should have an Input="1", because the signals will be *inputs* to this CDP component. That is, these signals will read their value from other CDP components in the system and provide them to the MSI server.

The signal created with the example above will get its value from the signal "Sinus.Output", because the Routing attribute was set to point to this signal.

Signals in the <Subscribe> section should have an Input="0" attribute, because they are *output* signals from the CDP component (other CDP components can read the signals).

3. Additional features

3.1. Publishing multiple objects

The MSI client supports publishing several objects. The (Marintek) MSI server which was used to test against during development did not support more than one object from each client. The documentation does not clearly state if more than one object from each federate should be allowed under the MSI protocol. To publish more than one object to servers that support only one object per client, one therefore needs to instantiate one MSI client for each object.

Version 2.0 update: The CSI server from Marintek running at OSC supports publishing several objects per federate.

3.2. TimeConstrained federate

The default behaviour of an application using the MSI client is to let every component run freely, without respect to the MSI tone signal. To allow one or more of the components in the application to be synchronized with the tone signal, special considerations have to be taken.

Normally, CDP components are scheduled periodically by the CDPEngine component. This means that messages delivery, state transitions and periodic methods happen automatically. To allow components to be synchronized with the tone signal from the MSI server, the components must be scheduled by the MSI client component instead. This may be achieved by following these steps:

- The component that should run time-constrained must have its priority set to “none” by setting the <Priority> elements value to “none” in the model file for the component.
- In the MSI component file, specify the TimeConstrained=”1” attribute inside the <Publish> element.
- In the MSI component file, set the name of the component to schedule by using the Component attribute inside the <Publish> element.

Example:

In Application.xml, instantiate a Thruster1 component of type “Thruster” and an MSI client:

```
...
<Components>
  <Component Name="Thruster1" src="Components\Thruster1.xml"></Component>
  <Component Name="MSI" src="Components\MSI.xml"></Component>
</Components>
```

To prevent the Thruster model(s) to run periodically, but to be scheduled by the MSI client, put this in the model file Models\Thruster.xml:

```
<Model>
...
<Priority>none</Priority>
...
</Model>
```

In the component file Components\Thruster1.xml set the requested frequency:

```
<Component Name="Thruster1" Type="Thruster">
  <Activate>1</Activate>
  <fs>10</fs>
...
</Component>
```

In the MSI component file, set attributes TimeConstrained and Component:

```

...
<Publish Object="StbdThruster" Type="Thruster" TimeConstrained="1" Component="Thruster1">
  <Signal Name="Force" Unit="N" Input="1" Routing="Thruster1.Force" Description="Starboard thruster force."></Signal>
</Publish>
...
    
```

This will publish an object “StbdThruster” with one signal “Force” to the MSI server. The CDP component Thruster1, running in the same application as the MSI client, will be scheduled by the MSI server as the tone signal from the server advances. Since fs was specified as 10Hz in the component file for Thruster1, the MSI client will ask the MSI server to run whenever the MSI server has advanced 0.1s, and it will call Thruster1's periodic Process...() method whenever it the tone has advances 0.1 s.

3.3. Messaging/interactions

In version 2.1 support for messages was added. To send a message to the CSI server, a *SendMessage* command must be sent to the MSI client. The parameter to the MSIMessage will be filled into the message.

For instance, if a MSIMessage with parameter:

```
name="ObjectName" speed="1.2"
```

is sent to the MSI client, it will produce the following message to the CSI server:

```
<message name="ObjectName" speed="1.2" />
```

To receive messages from the CSI server, the *name* attribute must match the name of a CDP component running in the same application as the MSI client. The MSI client will locate the CDPCOMPONENT by the name specified in the received message, create an *MSIMessage* and send it to the target component.

To process this message, the target component must therefore implement a handler for the *MSIMessage* command. Here is an example of a dummy handler for the MSIMessage command:

```

/**
Message sent via MSI server
*/
int MassWithTwoSprings::MessageMSIMessage(void* message)
{
    MessageTextCommandWithParmaReceive* pMsg = (MessageTextCommandWithParmaReceive*)message;

    if (DebugLevel(DEBUGLEVEL_EXTENDED))
        CDPMessage("%s: Received MSIMessage: %s", Name(), pMsg->parmas);

    // Code goes here

    return 1;
}
    
```

In addition, the MSIMessage must be added to the component, typically using the CDPDeveloper's Add Message command.

3.4. Debugging

To debug communication with the MSI/CSI server, go to the web page for the MSI client (typically <http://localhost/MSI>), and set the Debug property to a value between 0 and 9. Setting Debug to 9 should show all communication between the MSI client and the server (in the output window of the application).