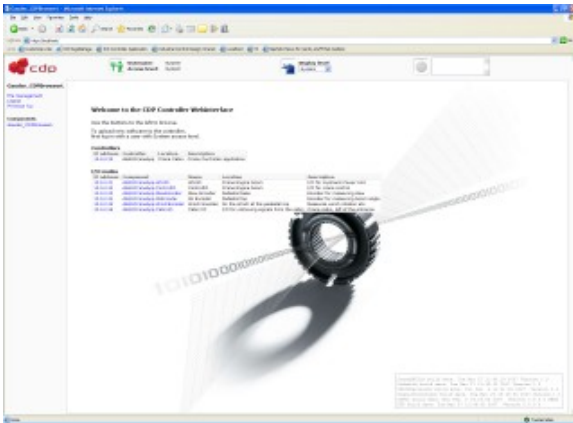




Product:	Web Server
Product version:	V2.3.1.0
Document ID:	UM-Web Server
Doc revision:	B
Written/Appr.:	SEL / RE
Date:	13. Oct. 2008

Industrial Control Design AS



Web Server 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The content of this document is confidential information not to be published without the consent of Industrial Control Design AS.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
1.1. About.....	3
<hr/>	
2. DESCRIPTION.....	3
2.1. Configuration.....	3
2.2. Standard GET requests.....	4
2.3. Server Side Scripting (SSS).....	4
2.4. Web Server Commands (HTTP GET requests with parameters).....	5
2.5. HTTP PUT.....	6
2.6. HTTP POST request.....	6

1. Introduction

1.1. About

The CDP WebServer provides web browser access to CDP controller applications and components on http:// protocol.

2. Description

2.1. Configuration

The component xml file of the webserver (usually WebServer.xml) contains the configuration parameters for the web server.

Here is an example of the file:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Default WebServer CDP system component. -->
<Component Name="WebServer" Model="WebServer">
  <NrConnections>10</NrConnections>
  <Activate>1</Activate>
  <NetworkInterface LocalName="ETH0"></NetworkInterface>
  <Parameters>
    <Parma Name="Port" Value="80" DefaultValue="80" PreviousValue="80" TimeLastChanged="Fri Oct 29 13:13:32 2004"
      Description="The port that the webserver listens to."></Parma>
  </Parameters>
</Component>
```

NrConnections

Number of threads that serve web server requests. (Maximum simultaneous requests).

Port

Port number used for web server. Default is standard http port 80.

2.2. Standard GET requests

When the web server receives a standard HTTP GET request, it will either return a file from the disk, or an automatically generated document, depending of the target and extension of the url. As a rule, the web server will return the file referenced by the URI, and the WebServer folder is the top-level folder on the controller. The exceptions are all paths that match the name of a component which exist on the controller (with the periods replaced by forward slashes).

Assuming there is a component called MyApp.Sinus and a file called SomeFile.jpg in the WebServer directory, the following table illustrates what will be returned.

http://addr/MyApp/Sinus

The html page for the component. If no custom page has been created for this component, the default component web page will be returned (Component.html). Note that the periods in the object name has been converted to forward slashes.

http://addr/MyApp/Sinus.xml The auto generated xml for the component.

http://addr/JustAnotherFile.xml The file JustAnotherFile.xml from the WebServer directory.

http://addr/ The page TopLevel.html from the WebServer directory.

http://addr/SomeFile.jpg The file SomeFile.jpg from the WebServer folder.

2.3. Server Side Scripting (SSS)

The CDP web server supports some simple Server Side Scripting (SSS). The scripts are enclosed in `<% the script %>` tags. The scripts can be used to perform web server commands or to insert auto-generated text into the page containing the script.

The web server will replace the SSS tags with the result from the script before the page is sent to the client.

Inserting auto-generated text into the html page

By using the format `<% Response.write($VARIABLENAME) %>` the web server will replace the SSS tag with the auto-generated text.

The following table lists the variables that the web server knows and their meanings.

Variable	What will be inserted
\$OBJECTNAME	The full name of the object which 'owns' the page being sent.
\$OBJECTSHORTNAME	The short name of the object which 'owns' the page being sent.
\$OBJECTPATH	The path of the object which 'owns' the page being sent. This is the same as the full name of the object with the dots replaced with forward slashes.
\$MODELNAME	The name of the model of the object which 'owns' the page being sent.
\$OBJECTXML \$COMPONENTXML	The path of the component xml file. This is the same as \$OBJECTPATH + ".xml".
\$OBJECTXSL \$COMPONENTXSL	The path of the component xsl file (currently 'Component.xsl').
\$CONTROLLERNAME	The component name of the controller (that is, the application object).
\$USERNAME	The user name of the user logged in from the ip-address that requested the page.
\$ACCESSLEVEL	The access level of the logged in user ('Read only', 'Read/Write' or 'System').
\$SIGNALVALUE(<i>signalShortName</i>)	Insert the current signal value of the component's signal <i>signalShortName</i> . Example, in DemoWinch.html: Current winch speed: <code><% Response.write(\$SIGNALVALUE(Speed)) %></code>
\$PARMAVALUE(<i>parmaShortName</i>)	Insert the parameter value of the component's parameter <i>parmaShortName</i> . Example, in DemoWinch.html: Winch inner diameter: <code><% Response.write(\$PARMAVALUE(innerDiameter)) %></code>
\$BUILDDATE(<i>parmaShortName</i>)	Insert the build date of packages.
\$DISPLAYLEVEL	Insert the current selected display level.

2.4. Web Server Commands (HTTP GET requests with parameters)

The CDP web server supports some special HTTP GET requests. The requests is encoded as a standard HTTP GET request with parameters, and can easily be generated by a html form or with a link.

Return value

On success, the web server will return HTTP status code 200 OK.

On error, a suitable HTTP status code will be returned, typically 401 Not authorized, 404 File not found or similar.

The general format is:

<http://controllerIP/objectPath?Command=commandName&ParamName=parmaValue>

Example:

<http://192.168.0.151/DemoWinch/Joystick?Command=SetSignal&SignalValue=0.2>

This table lists the commands that are directed at a specific object (the *objectPath* part of the URI is specified):

Command	Parameter	Description
Message=CreateComponent	ComponentType= <i>componentType</i> &ComponentName= <i>componentName</i>	Create a new CDP component (or object) of type <i>componentType</i> with the name <i>componentName</i> .
Message=CreateSignal	SignalType= <i>signalType</i> &SignalName= <i>signalName</i>	Create a new signal of type <i>signalType</i> with the name <i>signalName</i> .
Message=SetSignal	SignalValue= <i>value</i>	Set the signal specified by <i>objectPath</i> to the new value. Hexadecimal values may be specified by using '0x' prefix..
Message=SetSignal	SignalName= <i>signalName</i> &SignalValue= <i>value</i>	Set the signal <i>signalName</i> to <i>value</i> . Hexadecimal values may be specified by using '0x' prefix..
Message=SetParma	ParmaValue= <i>value</i>	Set new value for the CDPParma object specified by <i>objectPath</i> . Hexadecimal values may be specified by using '0x' prefix..
Message=SetParma	ParmaName= <i>parmaName</i> &ParmaValue= <i>value</i>	Set the CDPParma <i>parmaName</i> to <i>value</i> . Hexadecimal values may be specified by using '0x' prefix..
Message=AcknowledgeAlarm	AlarmName= <i>alarmName</i>	Parameter is optional. If specified, the alarm object <i>alarmName</i> is acknowledged. Otherwise, the object specified by <i>objectPath</i> is acknowledged.
Message=RouteSignal	OtherEnd= <i>otherSignal</i>	Route the signal specified by <i>objectPath</i> to the signal <i>otherEnd</i> .
Message= <i>cdpMessage</i>	&Parameter= <i>MyParameter</i> [not required]	Send the message <i>cdpMessage</i> as a CM_TEXTCOMMAND over the Messenger to the object specified by <i>objectPath</i> .
For instance: Message= SetProperty	PropertyName= <i>PropertyVal</i>	Only zero or one parameter is supported. If a parameter is specified, the WebServer sends the command using MessageTextCommandWithParmaSend(). If more data needs to be transferred, use HTTP POST instead (see chapter 2.6).
Message=CM_ACTIVATE	[no parameters]	For a list of messages a component supports receiving, see the component documentation, or look at the component web page, under the Messages section. To see all available messages, set DisplayMode to Debug in the top of the page.
Message=MyCustomMsg	<i>SomeOptionalString</i>	
Get=ModelDoc		Get the auto-generated documentation for the target's model Includes description of the state machine, signals, parameters, messages, alarms etc.
Message=CDPXMLList	Type=Signals &Page=1 &PerPage=2 &Start=1 &Filter=Pump &NMatching=3	Return result from calling CreateDynamicXML() on the destination component. This will return a partial list of signal, parameters, alarms or properties as specified by the parameters. This command is used by the web interface to retrieve data for instance when a user changes to the next page of Signals.

This table lists the commands that are directed at the web server (the objectPath part of the URI is not specified)

Command	Parameter	Description
Username= <i>username</i>	Password= <i>password</i>	Log in with new user name and password to set new access level.
Command=Logout	Name= <i>anything</i>	Log out the user that is currently logged in from the address that sent this command.
Command=Dir	Name= <i>dirName</i>	Return directory listing for the directory <i>dirName</i> .
Command=MakeDir	Name= <i>dirName</i>	Create the new directory <i>dirName</i> .
Command=RemoveDir	Name= <i>dirName</i>	Remove the directory <i>dirName</i> .
Command=RemoveFile	Name= <i>fileName</i>	Remove the file <i>fileName</i> .
Command=DownloadFile	Name= <i>fileName</i>	Download the file <i>fileName</i> . Note: The WebServer directory is not the top level folder when using this command. The top level folder of the current drive is.
Command=GetComponentList	Name= <i>componentName</i>	Return a xml document with hierarchical list of subcomponents of <i>componentName</i> in xml format. If <i>componentName</i> is 'All', a hierarchical list of all components is returned. If <i>componentName</i> is 'TopLevel', only the top level components is returned.
Command=GetApplicationList	Name= <i>something</i>	Return xml document with Messenger's list of applications with name, number, handle, ip-address and messenger port number, (the same list as can be seen by presing 'c' in the console window). Note that the dummy parameter Name= <i>something</i> is required (<i>something</i> can be anything).
Command=SetSignal	SignalName= <i>signalFullName</i>	Set new signal value for the signal .
Command=GetTimeStamp	Name= <i>Juks</i>	Get the build date of packages.
Command=GetLastMessages	Name= <i>Juks</i>	Get the message log (debug output messages)
Command=GetInfo	Name=UserAccess	Get the username and access level
Command=GetMac	Name= <i>anything</i>	Get the mac address of the controller

2.5. HTTP PUT

The Web server supports HTTP PUT.

The user performing the PUT request must have accesslevel System.

2.6. HTTP POST request

HTTP POST requests may be used to post data to a CDPObject. A POST will cause the web server to call the destination object's SetProperty() method. This enables sending a large block of data to an object.

In contrast, the *?Message=SomMessage* method described above will only allow sending approximately 1 kB of parameter data, limited by allowed length of uri (?) and max CDP message length (approximately 1 kB).

To use HTTP POST to send a message to an object, use */ComponentName* as action, and put the parameters to set in the body on the format PropertyName=PropertyValue, one parameter per line.

This could be accomplished for instance by the following HTML code:

```
<form method="POST" action="/ComponentName">
  <input type="text" name="Description">New description goes here</input>
  <input type="submit">Set description</input>
</form>
```

Only properties that are prepared for reception via HTTP POST can receive this command. At the time of writing that is only CDPComponent's Description property.

As an example on how to receive the POST request, here's how CDPComponent::SetProperty() has implemented the reception of the HTTP POST command for the *Description* property:

```

CDPComponent::SetProperty(std::string propertyName, std::string propertyValue)
.
.
.
if (propertyName=="HTTP POST")
{
  int eqSign = propertyValue.find_first_of('=');
  std::string propName = propertyValue.substr(0, eqSign);
  std::string propValue = propertyValue.substr(eqSign+1, propertyValue.size() - eqSign - 1);
  if (propName=="Description")
  {
    if (Application::GetAuthorizationLevel() >= Application::AccessLevelSystem)
    {
      m_descriptionComponent = (std::string("<![CDATA[" + propValue + "]]>");
      xmlComponent.SaveValue( propName.c_str(), m_descriptionComponent.c_str() );
      xmlComponent.SaveXMLFile();

      if (DebugLevel( DEBUGLEVEL_EXTENDED ))
        CDPMessage("%s.SetProperty(HTTP POST): %s=%.500s.\n", Name(), propName.c_str(), \
          propValue.c_str());
      return;
    }
    else
    {
      HTTPException httpException("%s.SetProperty(HTTP POST): Error: Need access level System\
to set %s property.\n", Name(), propName.c_str());
      httpException.SetErrorCode( HTTPEXCEPTION_UNAUTHORIZED );
      throw httpException;
    }
  }
  else
  {
    HTTPException httpException("%s.SetProperty(HTTP POST): Error: SetProperty %s not\
implemented for this component.\n", Name(), propName.c_str());
    httpException.SetErrorCode( HTTPEXCEPTION_BAD_PARAMETER );
    throw httpException;
  }
}
}

```