



***i*.LON™ 100 *e2* Internet
Server User's Guide:
Using the i.LON 100 Web Pages to Configure
Applications and to Monitor and Control
Data Points**

078-0289-01A

Echelon, LON, LONWORKS, LonTalk, LonBuilder, LonManager, Neuron, 3120, 3150, LONMARK, NodeBuilder, and the Echelon logo are trademarks of Echelon Corporation registered in the United States and other countries. LonMaker, LNS, and *i*.LON are trademarks of Echelon Corporation.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Echelon Corporation.

Printed in the United States of America.
Copyright ©2002-2004 by Echelon Corporation.

Echelon Corporation
550 Meridian Ave
San Jose, CA 95126, USA

Preface

This document describes how to use the *i.LON 100* Web pages to configure the *i.LON 100* applications and how to create custom Web pages using the SOAP interface.

Welcome

The *i.LON 100* Web pages can be used to configure the *i.LON 100* applications. Using the *i.LON 100* Web pages, you can configure the *i.LON 100* to perform alarming, scheduling, data logging, digital input and output, and pulse counting.

Purpose

The *i.LON 100 User's Guide: Using the i.LON 100 Web Pages to Configure Applications and to Monitor and Control Data Points* describes how to configure the *i.LON 100* application using the *i.LON 100* Web Pages and how to design web pages that can be used to monitor and control *i.LON 100* Data Points.

Related Documentation

i.LON 100 User's Guide: Installing, Connecting, and Configuring the i.LON 100 – Describes how to connect the *i.LON 100* Internet Server hardware and configure it to communicate by TCP/IP, LONWORKS messaging, email, and POP.

The *i.LON 100 User's Guide: Configuring the i.LON 100 Applications Using the i.LON 100 Configuration Plug-in* – Describes how to use the *i.LON 100* Configuration Plug-in to configure the *i.LON 100* applications.

LNS For Windows Programmer's Guide, xDriver Extension – Describes how the xDriver software can be used by an LNS application to manage communications with multiple LONWORKS networks that communicate over a TCP/IP network. The xDriver software is used to communicate with the *i.LON 100* when the *i.LON 100* is functioning as a Remote Network Interface (RNI).

LNS Programmer's Guide – Describes how to write LNS applications that can take advantage of the communication provided by the *i.LON 100* Web server.

LonMaker User's Guide – Describes how to use the LonMaker tool, which can be used to connect the *i.LON 100* functional blocks to a LONWORKS network.

i.LON 100 Programmer's Reference – Describes how to configure the *i.LON 100* using XML files and SOAP calls. This allows you to configure the *i.LON 100* without using the *i.LON 100* Configuration Plug-in

Software Requirements

To configure the *i.LON 100* using the *i.LON 100* Web pages, you must use Internet Explorer 6 or later.

Table of Contents

Preface	i
Welcome.....	ii
Purpose	ii
Related Documentation	ii
Software Requirements	ii
Table of Contents	ii
1 Introduction	1-1
<i>i.LON 100</i> Web Page Capabilities	1-2

<i>i</i> .LON 100 Web Page Limitations	1-2
Getting Started with the <i>i</i> .LON 100 Web Pages	1-2
Web Page Poll Rates	1-4
Using Web Binding	1-4
Configuring Internet Explorer For Web Binding	1-4
Creating a Web Binding Connection	1-5
2 Data Points	2-1
Data Points	2-2
Data Point Types	2-2
Data Point Presets	2-3
Configuring Data Points	2-4
Reorganizing the Data Point Tree	2-4
Modifying Data Point Properties	2-5
Adding and Removing Presets	2-6
Reading and Writing Data Points	2-7
3 Managing Alarms	3-1
Alarming Overview	3-2
The Alarm Generator Application	3-2
The Alarm Notifier Application	3-3
Configuring an Alarm Generator	3-3
Providing an Alarm Notification	3-6
Latching, Acknowledging, and Clearing Alarms	3-6
Using the Alarming Web Pages	3-7
The Alarm Summary Web Page	3-8
The Alarm History Web Page	3-10
4 Logging Data	4-1
Data Logging Overview	4-2
Creating a Data Logger	4-2
Adding Data Points to a Data Logger	4-4
Extracting Data Logs	4-5
Viewing Data Logs	4-6
5 Scheduling	5-1
Scheduling Overview	5-2
Creating a Schedule	5-2
Planning Out Your Schedule	5-3
Example of Planning Out a Schedule	5-3
Creating a Scheduler Application Instance	5-3
6 Using Digital Inputs and Digital Outputs	6-1
Digital Input Overview	6-2
Using a Digital Input	6-2
Digital Output Overview	6-3
Using a Digital Output	6-3
7 Using Pulse Counter Inputs	7-1
Pulse Counter Overview	7-2
Using the Pulse Counter	7-2
8 Creating a Web Page	8-1
Introduction	8-2
User Web Page Locations	8-2

Communicating with the DataServer	8-2
The Microsoft Web Service Behavior	8-3
Beginning Tutorials	8-3
Tutorial 1: Reading Data Points	8-3
Step 1: Create a list of data points	8-3
Step 2: View the Web Page	8-4
Tutorial 2: Customizing the Display	8-4
Step 1: Referencing HTML Objects	8-5
Step 2: Initializing the Display	8-5
Step 3: Creating Custom HTML Objects	8-5
Step 4: Displaying Data Point Values	8-6
Tutorial 3: Changing the Poll Rate	8-7
Step 1: Choosing a Default Poll Rate	8-7
Step 2: Reading User Input	8-8
Tutorial 4: Writing to Data Points	8-9
Step 1: Calling the writePoints Function	8-9
Step 2: Getting User Input	8-9
Step 3: Handling a Successful Response	8-10
Step 4: Handling Errors in the Response	8-10
Advanced Tutorials	8-10
Tutorial 5: Enabling SOAP in the Browser	8-11
Step 1: Attach the Microsoft DHTML Web Service Behavior to a Web Page	8-11
Step 2: Define Links to the WSDL File and SOAP Path	8-11
Step 3: Start the Web Service Behavior	8-12
Tutorial 6: Sending Low Level SOAP Messages	8-13
Step 1: Add a Button to the Web Page	8-13
Step 2: Create the Request String	8-13
Step 3: Send the SOAP Message	8-14
Step 4: Handle the Response	8-14
Tutorial 7: Converting XML to Native JavaScript Objects	8-15
Step 1: Creating a Request Object	8-15
Step 2: Converting the Response XML to Objects	8-17
Tutorial 8: Writing to Data Points	8-17
Step 1: Add a Button to the Web Page	8-18
Step 2: Add a Text Box to the Web Page	8-18
Step 3: Create the Request Object	8-18
Step 4: Send the Message	8-18
Step 5: Handle the Response	8-19
Tutorial 9: Using Presets – the UCPTvalueDef Element	8-20
Step 1: Adding a Text Box to the Web Page	8-20
Step 2: Reading and Displaying Data Point Presets	8-20
Step 3: Writing to Data Points Using Presets	8-21
Tutorial 10: Polling Data Points in JavaScript	8-21
Step 1: Start Reading Data Points	8-21
Step 2: Read the Time Stamp of the Response	8-21
Step 3: Start the Timer for the Next Read	8-22
Step 4: Copy the Time Stamp to the Next Message	8-22
Step 5: Stopping the Poll Task	8-23
Using Other Tools To Build <i>i.LON 100</i> Web Pages	8-23
9 <i>i.LON 100</i> User Web Page Security	9-1
Overview of <i>i.LON 100</i> User Web Page Security	9-2
Setting Access Restrictions	9-2
Users and Groups	9-3
Locations	9-5

Realms.....	9-5
Aliases	9-6
Parameters	9-7
Sample WebParams.dat file.....	9-7

A Creating a Web Page Using Web Tags A-1

Overview of Creating <i>i.LON 100</i> Web Pages Using Web Tags	A-2
Using the <i>i.LON 100</i> Server's Web Server	A-2
Required Hardware	A-2
Required Software	A-2
Creating The LonMaker Network	A-3
Creating Web Pages	A-3
How the HTML Code Works	A-5
Using The Web Server Functional Block.....	A-7
Required Hardware	A-7
Required Software	A-7
Setting Up The Hardware	A-7
Creating the LonMaker Network	A-7
Creating Web Pages	A-10
<iLonWeb> Web Tag Format	A-12
FUNC Attribute.....	A-13
Func=ShowValue	A-13
FUNC=Include	A-13
FUNC=CreateSymbol.....	A-14
SYMBOL Attribute.....	A-14
Data Point Symbols (NVL_, NVE_, and NVC_ Prefixes)	A-15
System Symbols (ILON_ Prefix).....	A-17
Web Tag Attributes	A-21
FIELD:	A-21
FORMAT:	A-22
Standard Resource File Set	A-22
User Resource File Sets.....	A-22
Built-in Formats	A-23
PROPAGATE:.....	A-23
WAIT:	A-23
Working with Forms	A-24
Opening a Form	A-24
Submit or Reset a Form	A-25
Refresh a Form	A-25
Form Element Functions	A-26
CheckBox	A-26
Hidden	A-27
RadioButton	A-27
TextField.....	A-28

B Troubleshooting B-1

1

Introduction

This chapter provides an overview of using the *i.LON 100* Web pages to configure *i.LON 100* applications.

***i*.LON 100 Web Page Capabilities**

The *i*.LON 100 Web pages can be used to configure the *i*.LON 100 applications from any computer running Internet Explorer 6 or better. The *i*.LON 100 Web pages have the following capabilities:

- The *i*.LON 100 Web pages are used to configure the *i*.LON 100 to communicate via TCP/IP, PPP, and LonWorks protocols, as well as to configure Security options. See the *i.LON 100 User's Guide: Installing, Connecting, and Configuring the i.LON 100* for more information about these capabilities.
- The *i*.LON 100 Web pages can be used to configure the *i*.LON 100 scheduling, alarming, data logging, digital input and output, and pulse counting applications. These applications can also be configured using the *i*.LON 100 Configuration Plug-in, as described in The *i.LON 100 User's Guide: Configuring the i.LON 100 Applications Using the i.LON 100 Configuration Plug-in*.
- The *i*.LON 100 Web pages can be used to read data log and alarm values and to acknowledge and clear alarms.
- The *i*.LON 100 Web pages can be used to read, write, and configure data points, and also to create and manage preset values for data points.
- Custom *i*.LON 100 Web pages can be created that allow data point values to be read and written using SOAP messages. This allows you to design a custom interface for *i*.LON 100 data that can be accessed by any computer running Internet Explorer 6 or better.

***i*.LON 100 Web Page Limitations**

The *i*.LON 100 Web pages have the following limitations:

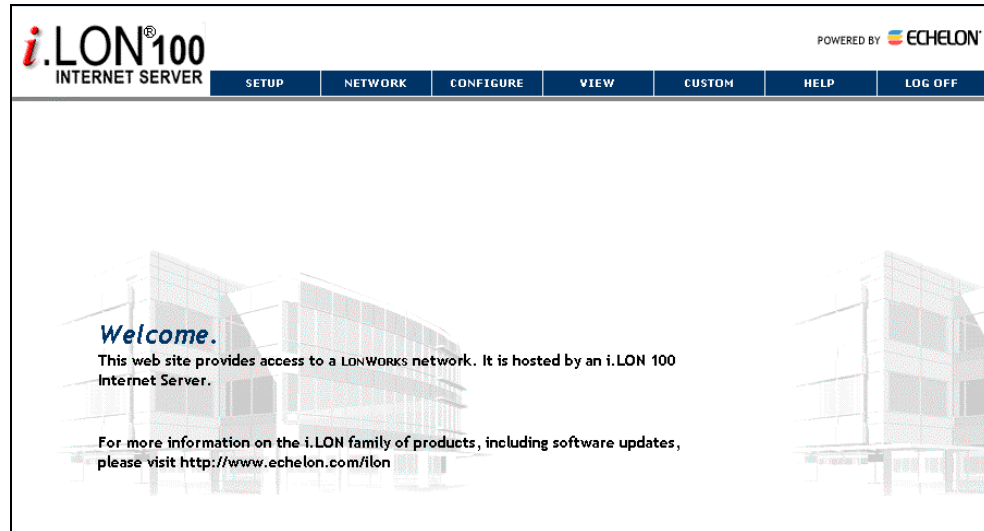
- The *i*.LON 100 Web pages cannot be used to configure the **Type Translator** or **Analog Function Block** applications. To configure these applications, use the *i*.LON 100 Configuration Pug-in.
- The *i*.LON 100 Web pages cannot be used to create NVE or NVL data points. NVE data points can be created using the *i*.LON 100 Configuration Plug-in. NVL data points are created by adding dynamic network variables to the *i*.LON 100, which can be done with an LNS tool such as the LonMaker tool.
- The *i*.LON 100 Web pages cannot be used to change network variable types on *i*.LON 100 functional blocks. This means that the type of data sent by the **Pulse Counter** application, for example, cannot be changed using the Web page interface.

Getting Started with the *i*.LON 100 Web Pages

To access the *i*.LON 100 Web pages, point Internet Explorer 6 or better to the IP address or hostname of your *i*.LON 100. See *i.LON 100 User's Guide: Installing, Connecting, and Configuring the i.LON 100* for more information on configuring IP information.

This opens the *i*.LON 100 Web startup page. The startup page describes the various *i*.LON 100 applications. Before proceeding, you should note that the *i*.LON 100 Web page interface uses pop-ups. If you have any pop-up blocking software, close it before accessing the *i*.LON 100 Web pages.

Click **Login** to open the *i.LON 100* Welcome Web page and begin configuring the *i.LON 100* applications. You will be prompted for a user name and password. By default, this is *ilon/ilon*. Once you supply the user name and password, the *i.LON 100* Welcome Web page appears as shown in the following figure:

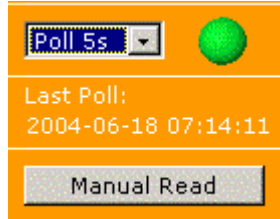


This page contains the following 7 menus:

- | | |
|------------------|--|
| Setup | This menu is used to access Web pages used to configure IP, Modem, Time Server, and Security information. See the <i>i.LON 100 User's Guide: Installing, Connecting, and Configuring the i.LON 100</i> for more information on these Web pages. |
| Network | This menu is used to access Web pages used to create LAN/WAN and LonWorks connections, and to communicate with M-Bus devices. See the <i>i.LON 100 User's Guide: Installing, Connecting, and Configuring the i.LON 100</i> for more information on these Web pages. |
| Configure | This menu is used to access Web pages used to configure the <i>i.LON 100</i> Alarming, Scheduling, Data Logging, Digital Input/Output, and Pulse Counting applications, as well as to configure data points and Web binding (the process by which an <i>i.LON 100</i> can be connected to another <i>i.LON 100</i> or a third party server). These Web pages are described in this document. |
| View | This menu is used to access Web pages used to view data logs, alarm logs, and data point values. |
| Custom | Click Custom to open <i.LON 100 URL>/user/index/html. This location on the <i>i.LON 100</i> is reserved for custom Web pages. |
| Help | Click Help to display help for the current Web page. |
| Log Off | Click Log Off to log off of the <i>i.LON 100</i> . |

Web Page Poll Rates

All of the *i*LON 100 application configuration Web pages and the alarming, data log, and data log view Web pages allow you to set the polling rate for the Web page. On the left side of these Web pages, the polling options appear, as shown in the following figure:



You can use the drop-down menu to determine how often, in seconds, the *i*LON 100 is polled to update the data on this Web page. Set this option to **Manual** to perform polling only when the **Manual Read** button is clicked. The color of the circle to the right of the drop-down menu indicates the polling status:

- | | |
|--------------|--|
| Gray | Polling is currently stopped. |
| Green | The last polling attempt was successful. |

Below the drop-down menu the time of the latest successful poll is displayed under **Last Poll**. Click **Manual Read** for force the Web page to poll the *i*LON 100 immediately; when you do this the drop-down polling menu will be set to **Manual**.

Using Web Binding

The *i*LON 100 can connect data points directly to data points on another *i*LON 100 or to a third party web server such as Apache® or IIS® by a process called Web binding, which uses SOAP messaging as a transport. No connection with an LNS Server is required, and *i*LON 100 devices connected via Web binding do not need to be in the same LNS database; Web binding connections are independent of LonWorks domain boundaries. Web binding does not tightly constrain the types of the data points involved in a connection, which enables simple translation for scalar to scalar connections. For example, even though the underlying data types of **SNVT_temp_f** and **SNVT_temp_p** may be different, Web binding allows data points of these types to be connected and carries out the conversion automatically. Furthermore, structured types such as **SNVT_lev_disc** and **SNVT_switch** may be connected by using Presets to map values such as **ST_ON** and **100.0 1** both to a preset value of **ON**; see *Data Point Presets* in Chapter 2 for more information.

When the data point on the source *i*LON 100 is updated, the *i*LON 100 will send a request attempting to update the data point on the destination *i*LON 100 to which it is connected. No connection with an LNS Server is required.

Configuring Internet Explorer For Web Binding

When connecting two *i*LON 100 devices using Web binding, it is often convenient to see the data point trees on both devices. Internet Explorer security settings may prevent viewing of data from a another server even though you may be able to reach that server in a separate browser instance. To enable the viewing of multiple *i*LON data point trees in a single instance of Internet Explorer, you should follow these steps:

1. Start Internet Explorer.
2. Open the **Tools** menu and select **Internet Options**. The **Internet Options** dialog opens.
3. Select the **Security** tab.
4. Under **Select a Web Content Zone to Specify Its Security Settings**, select **Trusted Sites**, and then click **Sites**. The **Trusted Sites** dialog opens.
5. Clear the **Require Server Verification (https:) For All Sites In This Zone** option.
6. Enter the IP address (or host name + domain suffix if using DNS) for an *i.LON 100* that you plan to connect using Web binding using the format `http://123.123.123.123` (or `http://ilon-host-name.mydomain.com`), and then Click **Add**. Repeat this step for every *i.LON 100* you will connect using Web binding.
7. Click **OK** in all open dialogs, close all instances of Internet Explorer, and then start a new Internet Explorer session to enable the new settings.

Creating a Web Binding Connection

In order to connect data points on an *i.LON 100* to data points on one or more other servers, follow these steps:

1. Open Internet Explorer and point it to your *i.LON 100*.
2. Create one or more servers with the **Web Binder Destination** service as described in the *i.LON 100 User's Guide: Installing, Connecting, and Configuring the i.LON 100*.
3. Under **Configure**, select **Web Binder**. The following Web page opens:

On the left side of this page, under **Select a Source Data Point** a tree view of the *i.LON 100* appears. On the right side of this page, under **Select Destination Data Points**, tree views of all available Web binder destinations appear; this tree will always contain a **Turnaround Address** which is used to bind an *i.LON 100* to itself. You can add additional destinations from this page by clicking **Add Destination** in the right sidebar; this opens a window that requires you to fill in the same information you would need to create a Web binder destination service as described in the *i.LON 100 User's Guide: Installing, Connecting, and Configuring the i.LON 100*.

4. Navigate the *i*.LON 100 in **Select a Source Data Point** and select the data point you want to bind.
5. Navigate the **Select Destination Data Points** tree and select one or more data points to be bound to. If the destination server is not reachable create a new data point by selecting the *****New Data Point***** node.
6. Once you have selected a source data point and one or more destination data points, click the **Add Binding** button on the left sidebar. The new Web binding appears at the bottom of the window.
7. Once you have created one or more Web bindings, click **Validate** to validate that the web binding is valid. The **Web Binder Validation Report** window opens. This window shows all web bindings on the *i*.LON 100 device. For each binding, it shows a **Status**. If there is some reason that the web binding is invalid (can't find the IP address, format mismatch, etc), it will be shown here.
8. Click **Submit**.

To delete bindings from the table, select one or more bindings and then click **Delete Bindings**.

Note: Creating a Web binding from a version 1.1 *i*.LON 100 to a version 1.0 *i*.LON 100 is not supported.

Note: Once a dial-up connection is established, it will not be released until its **Disconnect If Idle For** time is reached. This means that if data is being constantly sent over a dial-up connection, the connection will never be dropped and any data that needs to use a second dial-up connection may never be set. If multiple source data points of Web binding connections use different dial-up connections, you must ensure that the source data points are not updated so frequently that the first PPP connection is never dropped. If data is being sent over a dial-up connection at a faster rate than the timeout for the connection, the connection will never be dropped, and a new connection can never be made. This can result in a situation in which the *i*.LON 100 will be unable to update a Web binding connection over a second dial-up connection. See the *i.LON 100 User's Guide: Installing, Connecting, and Configuring the i.LON 100* for more information on PPP connections.

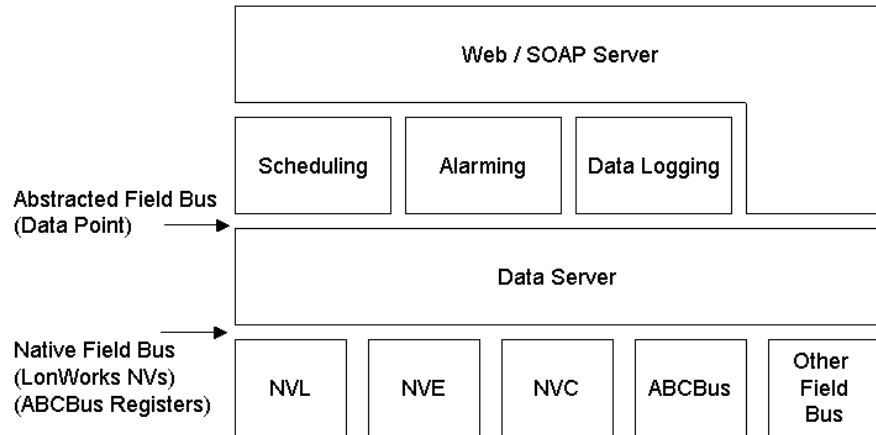
2

Data Points

This chapter describes how to configure, read, and write data points using the *i.LON 100* Web pages, as well as how to add and remove M-Bus data points.

Data Points

The *i*.LON 100 server's functional blocks and supporting applications are structured as diagrammed in the following figure:



Those familiar with other LONWORKS products are accustomed to thinking in terms of network variables. The *i*.LON 100 works with network variables, and also works with data elements from other field busses. For example, an *i*.LON 100 **Scheduler** functional block can schedule an M-Bus register just as easily as it can schedule a LonWorks network variable.

This flexibility uniquely positions the *i*.LON 100 to integrate legacy devices from other field busses. (Release 1.1 of the *i*.LON 100 applications includes NVL, NVE, NVC, and M-Bus drivers. Please contact Echelon support (lonsupport@echelon.com) directly for information on other third-party field bus drivers.)

The integration of other field busses with a LonWorks network is accomplished by the *i*.LON 100's data server. The data server is a software component that **abstracts** any data element of any bus into a **data point**. The *i*.LON 100's functional blocks (**Scheduler**, **Data Logger**, **Alarm Generator**, etc.) operate on data points – not network variables. Generally speaking you can consider a data point to be the same thing as a network variable because a network variable **is** a LONWORKS data point.

The *i*.LON 100 server can support up to 800 data points.

Data Point Types

The *i*.LON 100 server firmware supports the following data points types: *local data points* (also called NVL data points), *external data points* (also called NVE data points), *constant data points* (also called NVC data points), and *Meter Bus data points* (also called M-Bus data points):

Local data points correspond to network variables that are defined locally on the *i.LON 100* server. This includes any default network variables on an *i.LON 100* functional block, and any dynamic network variables you might add to an *i.LON 100* functional block. Local data points must be bound to one or more network variables on other devices in order to send or receive information on the LONWORKS network. Local data points can be created using the *i.LON 100* Configuration Plug-in.

External data points correspond to network variables on other LONWORKS devices. For those familiar with the *i.LON 1000* Web server, these are identical in concept to NVE tags, however the *i.LON 100* server abstracts the implementation as described above so the syntax for NVEs on the *i.LON 100* server differs from the syntax used on the *i.LON 1000* server.

The *i.LON 100* server's NVE driver keeps an XML file that contains all the information required to read and write external data points. This XML file is updated when you create NVE points using the *i.LON 100* Configuration Plug-in. You can also write to this XML file manually; see the *i.LON 100 Programmer's Reference* for more information.

Because NVE points are explicitly polled or poked they consume no network variable resources on the referenced devices but often at the expense of increased network traffic. External data points can be created using the *i.LON 100* Configuration Plug-in.

Constant data points are not associated with a network variable and are used to hold constant values. Constants are useful when making comparisons (for example, testing for alarm conditions) and when you need to supply a static value to some other device on your network. You can change the value of these data points using a custom Web page, using the SOAP/XML interface as described in the *i.LON 100 Programmer's Reference*. Local data points can be created using the *i.LON 100* Configuration Plug-in.

Meter Bus data points are used to communicate with Meter Bus (M-Bus) devices using the M-Bus protocol (EN 1434-3). M-Bus devices are connected to the *i.LON 100* via the Serial Port. For more information on creating M-Bus data points, see *The i.LON 100 User's Guide: Installing, Connecting, and Configuring the i.LON 100*.

Data Point Presets

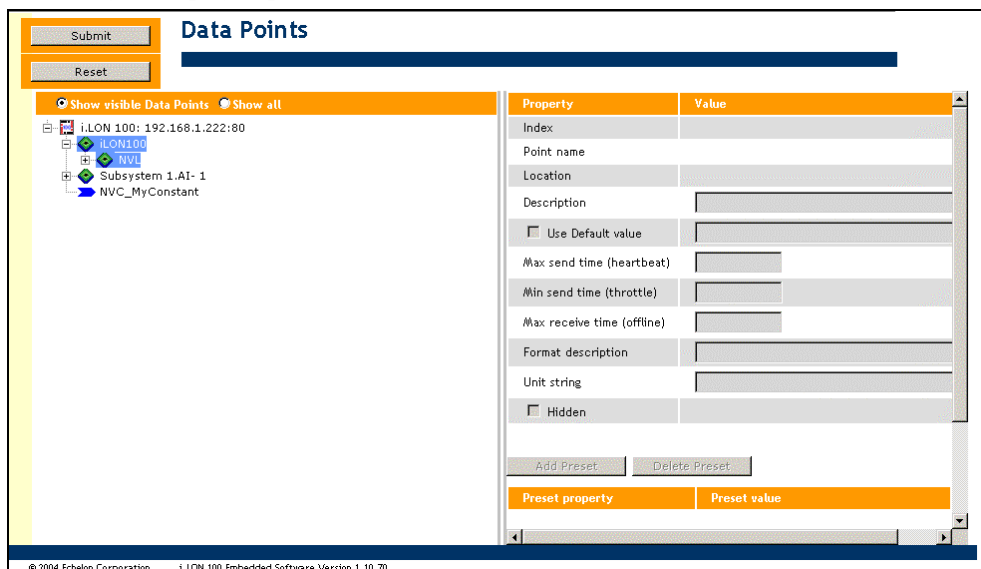
The *i.LON 100* server allows you to define presets for each data point. For example you might define a preset named **ON** for an `NVE_lampSw` data point (which is defined as a `SNVT_switch`) as 100.0 1. You might also define a preset named **ON** for an `NVL_heat_setpoint` data point (which is of type `SNVT_temp_f`) to be 22.

In both cases what you are saying is that you want to turn something **on** (lights or the heater), but the underlying data type needs to use 100.0 1 for lights and a floating point value of 22° C for the HVAC system. The data server in the *i.LON 100* server allows you to abstract the idiosyncrasies of the data types and use a mnemonic (ON) that makes sense to a human. This makes it much easier to work with the point later on. Whenever you want to drive an output network variable to a pre-defined value you set the data point to a pre-defined preset. For example, to turn the lights on at 8:00AM a scheduler block sets the data point `NVL_lampSw` to the ON preset, and the data server automatically translates ON to 100.0 1 as it updates the output network variable.

Another advantage of presets is that you can drive multiple outputs of different type simultaneously. For example, a scheduler can turn a group of points “ON” at 8:00AM, and if you included both the data points discussed above in the scheduled group the *i*LON 100 server would send the value 22 on NVE_heat_setpoint at 8:00AM and 100.0 1 on NVL_lampSw at 8:00AM. This is an extremely powerful feature that allows the *i*LON 100 server’s built-in applications to work with any kind of data type and field bus. This provides a layer of abstraction in the user interface so the end user does not need to know about the underlying data structures of the various data points.

Configuring Data Points

The *i*LON 100 Web pages can be used to configure existing data points on the *i*LON 100. To configure *i*LON 100 data points, hover your mouse cursor over the **Configure** menu and select **Data Points** from the drop down list. The following Web page opens:



Select a data point to view and modify its configuration. You can select multiple data points by holding down the <Ctrl> or <Shift> keys (just like Windows Explorer); if multiple data points are selected, any changes made to the data point properties on the right will be made for all selected data points.

Reorganizing the Data Point Tree

On the left side of the Web page is a tree that contains all the data points on the *i*LON 100 device. The tree is organized into:

Locations (📁) – Locations are analogous to folders in Windows Explorer. Locations can contain data points and other locations. Every location has a ‘+’ symbol next to it that can be clicked to expand the location.

Data points (📄) – Data points are analogous to files in Windows Explorer. Data points correspond directly to data points on the *i*LON 100.

You can perform the following reorganization operations upon this tree:

- **Create a Location** – You can create new locations to organize the data points on the *i*LON 100 device. Right-click any location in the tree and select **New Location** from the short-cut menu. A pop-up window will prompt you for the

name of the new location. Enter a name and click **OK** to have the new location added beneath the selected one. If a location does not have at least one data point in it, it will be removed when the browser is closed.

- **Cut/Paste** – You can cut and paste individual data points or entire locations. To cut and paste, follow these steps:
 - i.* Click a data point or location to select it. Selecting a location automatically selects all sub-locations and data points for that location. You can click multiple data points and locations to select all of them. To unselect a selected data point or location, click it again.
 - ii.* Once you have selected one or more data points and locations, right click one of the selected items and select **Cut** from the short cut menu. The selected items will become transparent.
 - iii.* Right-click a location and select **Paste** from the drop-down menu. All items selected in step *i* will be moved to the selected location.
- **Rename** – You can rename a location by right-clicking it and selecting **Rename** from the shortcut menu. A pop-up window will prompt you for the new name of the location.

Once you have finished reorganizing the data point tree, click **Submit** to save changes. The modified data point tree will now appear on the **Configure** and **Browse** Web pages.

Modifying Data Point Properties

To modify data point properties, follow these steps:

1. Click a data point in the data point tree to select it. You can select multiple data points by holding down the <Ctrl> or <Shift> keys just like Windows Explorer.
2. Modify the properties on the right-hand side of the page. The following properties are available for each data point:

Description	The description field for the selected data point(s). This field can contain up to 227 characters. This value has no effect on the function of the data point and is used to input description information.
Default Value	The default value for the data point.
Use Default Value	If this option is set, the data point will be set to the Default Value when the <i>i</i> .LON 100 reset. If this option is not set, the data point value will be set to 0 and the status will be set to AL_NUL when the <i>i</i> .LON 100 is reset.
Max Send Time (Heartbeat)	The maximum time between data point updates from the <i>i</i> .LON 100. If this amount of time elapses, an update will be sent even if the value has not changed.
Min Send Time (Throttle)	The minimum time between data point updates from the <i>i</i> .LON 100. Setting this property reduces network traffic by limiting the number of updates that are sent. If the value changes in less than this time, no update will be sent. The <i>i</i> .LON 100 will not

	automatically send the update when the Min send time timer has expired; for this reason it is recommended you also set the Max send time so the update is eventually sent.
Max Receive Time (Offline)	If a data point associated with an input network variable receives no input in this amount of time, the status will be set to AL_OFFLINE. This property applies to input local data points and all external data points (i.e. all points read by the i.LON 100). Set this property to 0 to disable maximum receive time.
Format Description	A description of the format of the data point. By default, this description is taken from the type file that defines the format. For NVL data points, this property is read-only. If you select multiple data points, you cannot modify this property.
Unit String	The units associated with the data point format. By default, this description is taken from the type file that defines the format.
Hidden	Set this option to hide this data point when browsing through i.LON 100 data points. You can see hidden data points by selecting the Show All option that appears above the tree view of the i.LON 100 data points on the left side of the Web page.

3. Click **Submit** to save changes.

Adding and Removing Presets

See *Data Point Presets*, earlier in this chapter, for more information about presets. To add a preset using the **Configure** Web page, follow these steps:

1. Click a data point in the data point tree to select it. You cannot create presets for multiple data points
2. Click the **Add Preset** button on the right side of the Web page. A new preset appears in the lower right corner of the Web page. Each preset consists of a **Preset Property** and a **Preset Value**.
3. Under **Preset Property**, enter the name for the new preset (this is the name by which i.LON 100 applications will refer to the new preset).
4. Under **Preset Value**, enter the value of the new preset. This value should be a legitimate value for the selected data point.
5. Click **Submit** to save changes.

To remove a preset, select it and click **Delete Preset**.

Default presets have been chosen for some network variable types so new data points that use one of these types will automatically have presets defined. For example, a **SNVT_switch** type data point will have presets for **ON** (100.0 1) and **OFF** (0.0 0) by default. These defaults may be changed by modifying the following file: `/root/config/software/dataserver/DPT_Preset.xml`

Reading and Writing Data Points

You can read and write data point values using the **View Data Points** Web page. To open this Web page, hover your mouse cursor over the **View** menu, and then select **Data Points**. The following Web page opens:

Index	Pri	Location
<input type="checkbox"/> 0	255	
<input type="checkbox"/> 6	255	iLON100/NVL/static/NodeObject/DeviceAlarm
<input type="checkbox"/> 7	255	iLON100/NVL/static/NodeObject/iLON-Location
<input type="checkbox"/> 5	255	iLON100/NVL/static/NodeObject/IP-Address
<input type="checkbox"/> 0	255	iLON100/NVL/static/NodeObject/Request
<input type="checkbox"/> 1	255	iLON100/NVL/static/NodeObject/Status
<input type="checkbox"/> 32	255	iLON100/NVL/static/RTC/StartSummerTime
<input type="checkbox"/> 33	255	iLON100/NVL/static/RTC/StartWinterTime
<input type="checkbox"/> 28	255	iLON100/NVL/static/RTC/TimeValue

On the left side of the Web page is a tree that contains all the data points on the *iLON 100* device. This tree can be reorganized as described in *Reorganizing the Data Point Tree*, earlier in this Chapter.

To view or modify a data point value, you must add the data point to the list on the right side of this Web page. To do this, follow these steps:

1. Click a data point in the data point tree to select it. You can select multiple data points by holding down the <Ctrl> key or by selecting a location to select all data points beneath it.
2. Click **Add Points**. The selected data points will be added to the list on the right side of the Web page.
3. The **Value** field shows the current value for each data point in the list. To modify the value, modify this field and click **Submit**.

To remove a data point from the data point list, set the checkbox to the left of the **Index** column and click **Delete Points**. This will remove the data points from the data point list on the right side of the Web page, not from the data point tree on the Left side of the Web page.

3

Managing Alarms

This chapter describes how to use the *i.LON 100* Alarming Web pages to define, read, and clear alarms.

Alarming Overview

The *i*.LON 100 Internet Server contains two applications that control alarming—the **Alarm Generator** and the **Alarm Notifier**. The *i*.LON 100 can use the **Alarm Generator** functional blocks to monitor up to 40 data points (one per functional block) and trigger alarms by setting the status of an input data point when specified conditions are met. For LONMARK integration with other alarming devices, the alarm generator can optionally update a **SNVT_alarm** and/or **SNVT_alarm_2** data point.

The **Alarm Notifier** functional blocks monitor the status of data points as well as conditions sent via **SNVT_alarm** and **SNVT_alarm_2** data points. **Alarm Notifier** functional blocks can be configured to respond to an alarm by updating one or more data points, sending email, and/or logging the alarm. A single **Alarm Notifier** functional block can be configured to monitor multiple inputs.

The Alarm Generator Application

The *i*.LON 100 includes 40 **Alarm Generator** application instances. An alarm generator generates alarms based on the values of any of the *i*.LON 100 data points. The alarm generator compares the values of an *input data point* with a *compare data point* each time either one is updated. You will select the function the alarm generator will use to make the comparison. If the result of the comparison is true, an alarm will be generated, and the status of the input data point will be updated to an alarm condition.

For example, you could select a Greater Than comparison function. The alarm generator generates an alarm when either data point is updated and the value of the input data point is greater than the value of the compare data point. The alarm generator includes the following *binary* comparison functions: Less Than, Less Than or Equal, Greater Than or Equal, Equal, and Not Equal.

The alarm generator also includes an *analog* comparison function. When you select this comparison function, you will specify four offset limits for the alarm generator. The four offset limits allow you to generate alarms based on how much the value of the input data point exceeds, or is exceeded by, the value of the compare data point. The alarm generator generates an alarm when either data point is updated and the difference between their values exceeds any of the offset limits.

You will define a hysteresis level for each alarm-offset limit when you use the analog comparison function. After an alarm has been generated based on an offset limit, the value of the input data point must return to the hysteresis level defined for that offset limit before the alarm clears, and before another alarm can be generated based on that offset limit. As a result, the alarm generator will not generate an additional alarm each time the input data point is updated after it reaches an alarm condition, but before it has returned to a normal condition.

All of the comparison functions have additional features that will allow you to throttle alarm generation. You can specify an interval that must elapse between alarm generations for a data point. You can also define an interval that must elapse after an alarm has returned to normal status before that alarm will be cleared. These features prevent the alarm generator from triggering multiple alarms each time the input data point reaches an alarm condition.

You can optionally select up to two *alarm data points* for each alarm generator, one of type **SNVT_alarm** and one of type **SNVT_alarm_2**. The status of these

data points will be updated to an alarm condition each time the alarm generator state changes (i.e. passive to active *or* active to passive).

The Alarm Notifier Application

The *i*.LON 100 includes 40 **Alarm Notifier** application instances. An alarm notifier logs alarm conditions and generates email messages and data point updates each time an alarm condition occurs. You can use an alarm notifier to provide notification of alarms generated by any device that produces a SNVT_alarm or SNVT_alarm_2 output, including the *i*.LON 100 device itself. For example, you can use an alarm notifier to provide alarm notification of alarms produced by alarm generators on the *i*.LON 100 device.

You will specify a group of input data points for each alarm notifier. The alarm notifier reads the status of these data points each time they are updated to determine if the alarm condition for the point has been changed to a value other than AL_NO_CONDITION (you can limit this further using the **Alarms** tab, as described below). If the data point has such a condition, the alarm is classified as an *active alarm* and an alarm notification is generated. Each time an input data point is updated and the alarm condition is set to AL_NO_CONDITION, the alarm is classified as a *passive alarm*.

You can specify one or more output data points for each alarm notifier. These data points will be updated each time an alarm notification occurs. You can also specify an email profile for each alarm notifier. An email message will be sent to the addresses specified for that email profile each time an alarm notification occurs. You can specify the message text, subject heading, and attachment to be included with each email notification. Email profiles allow you to notify different people when different alarms occur. This is useful if different groups of people need to receive notifications about the various alarm conditions that can occur on your network.

Each alarm notifier generates a log file. It will add an entry to this log file each time it causes an alarm notification. You can find these log files in the /root/AlarmLog directory of the *i*.LON 100 device. These files are named histlogX, where X represents the index number assigned to the alarm notifier when it was created. An alarm notifier will not generate a log file until it has generated an alarm notification.

The *i*.LON 100 device does not limit how much alarm data can be logged. However, you should maintain at least 1024KB of free disk space. You can view the amount of free disk space using the System Info Web page.

In addition, the alarm notifier generates a summary log that summarizes the log entries made by all alarm notifiers that were classified as active alarms. This file is called sumlog0, and can also be found in the /root/AlarmLog directory of your *i*.LON 100 device.

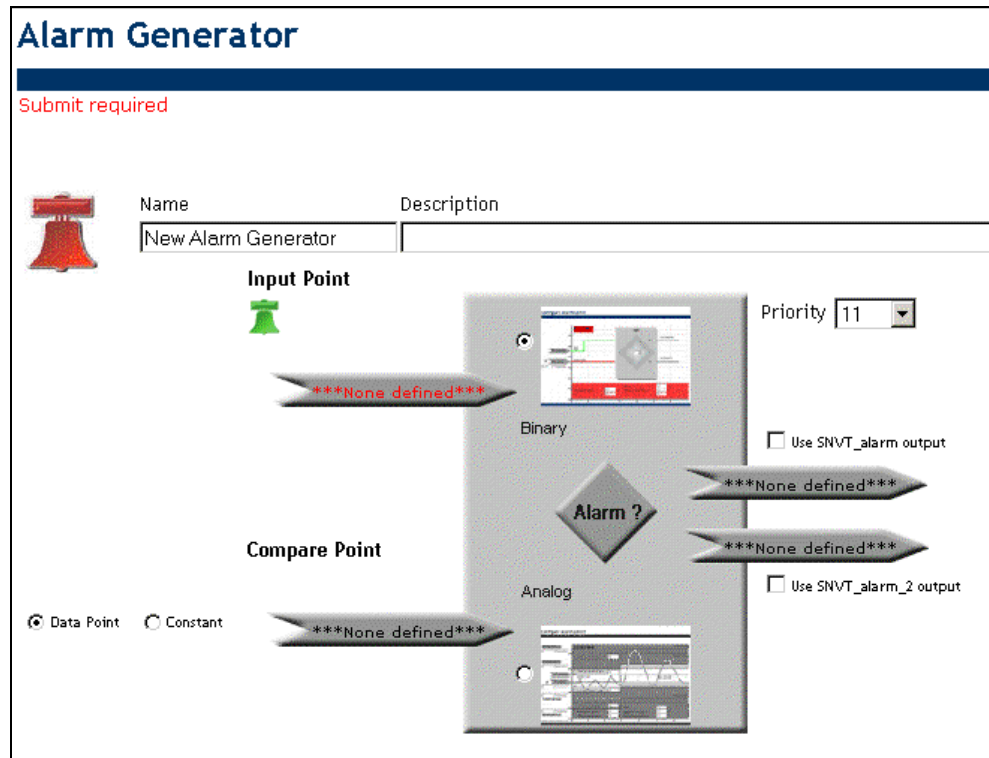
Configuring an Alarm Generator

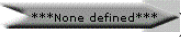
You can configure an **Alarm Generator** functional block to generate an alarm update in response to input conditions that you define. An alarm update does not result in an alarm notification. To provide notifications of an alarm update, connect the output of an alarm generator to one or more alarm notifiers (see ***Error! Reference source not found.***, later in this chapter).

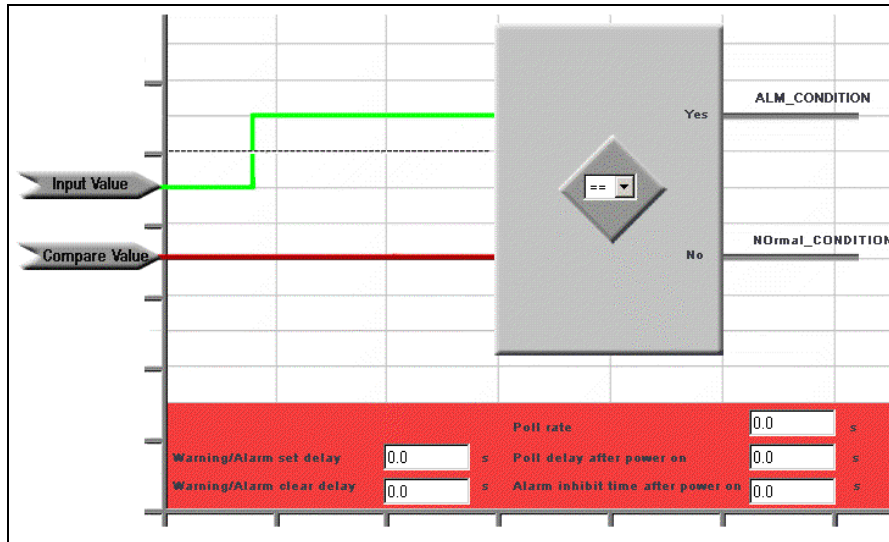
You can configure an Alarm Generator functional block using either the *i*.LON 100 Configuration Plug-in or the **Alarm Generator Configuration** Web page.

To configure an alarm generator to generate alarms using the *i*LON 100 Web pages, follow these steps:

1. Open Internet Explorer 6 or later, point it to the URL of the *i*LON 100, and log in to the *i*LON 100.
2. On the *i*LON 100 Web page, hover your mouse cursor over **Configure** and select **Alarm Generator** from the drop-down menu. The **Alarm Generator Configuration** Web page appears.
3. Click the **Add Alarm** button. A new **Alarm Generator** will be created as shown in the following figure:

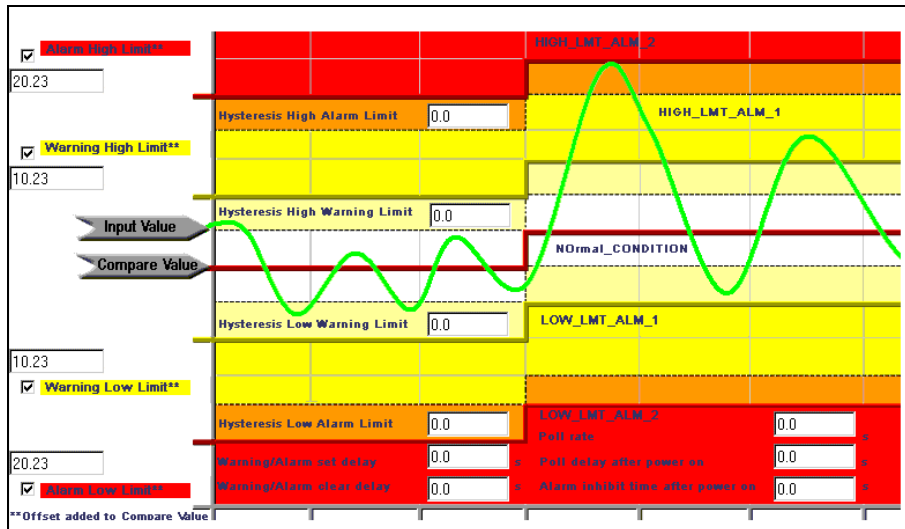


4. Enter the **Name** and optional **Description** for the new **Alarm Generator**.
5. Select an **Input Point** and a **Compare Point** for the new **Alarm Generator**. To select a point, click the data point icon () corresponding to the point to be set and select the data point from the tree that appears on the left side of the Web page. These data points must have the same type and format.
6. Optionally check the **Use SNVT_alarm Output** or **Use SNVT_alarm_2 Output** options. If either of these options is checked, you must click the associated data point shape and select a data point on which the alarm values will be sent.
7. Click the **Alarm?** button.
8. If you chose the *binary* option, the following window opens:



choose the comparison function to be used from the drop down menu. Available functions are equals (==), is not equal to (≠), greater than (>), greater than or equal to (>=), less than (<), and less than or equal to (<=). When the statement *Input Point (comparison) Compare Point* evaluates as true, an alarm will be generated.

If you chose the *analog* option, the following window opens:



Set the limits for this alarm. You can disable any of these limits by clearing the associated check box. For each limit, you can also set a hysteresis value (click **Help** for more information). The following limits can be set:

Alarm High Limit

When the **Input Value** exceeds the **Compare Value** by this amount or greater, a HIGH_LMT_ALM_2 condition will be generated. This value must be greater than the **Warning High Limit** value.

Warning High Limit

When the **Input Value** exceeds the **Compare Value** by this amount or greater, a HIGH_LMT_ALM_1 condition will be

Warning Low Limit

generated. This value must be lesser than the **Warning High Limit** value.

When the **Compare Value** exceeds the **Input Value** by this amount or greater, a LOW_LMT_ALM_1 condition will be generated. This value must be lesser than the **Warning High Limit** value.

Alarm High Limit

When the **Compare Value** exceeds the **Input Value** by this amount or greater, a LOW_LMT_ALM_2 condition will be generated. This value must be greater than the **Warning Low Limit** value.

9. Click **Submit** to save the new **Alarm Generator**.

Providing an Alarm Notification

You can configure an **Alarm Notifier** functional block to provide a network or email notification of an alarm condition. The alarm condition may be generated by an **Alarm Generator** functional block, it may be generated by another LONWORKS device that produces SNVT_alarm or SNVT_alarm_2 alarm outputs, or it may be generated in response to any data point going offline. To configure an alarm notifier to provide alarm notifications, follow these steps:

1. Open Internet Explorer 6 or later, point it to the URL of the *i*.LON 100, and log in to the *i*.LON 100.
2. On the *i*.LON 100 Web page, hover your mouse cursor over **Configure** and select **Alarm Notifier** from the drop-down menu. The **Alarm Notifier Configuration** Web page appears, as shown in the following figure:

The screenshot shows the 'Alarm Notifier Configuration' web page. On the left, a tree view shows 'Alarm Notifier-0' selected. The main content area includes a 'Description' text box, 'Delay time' (0 minutes, 0 seconds), 'Email aggregation time' (0 milliseconds), 'Logs' section with 'Format' set to 'Binary', 'History Log' (Entries: 0, Size (KB): 0), and 'Summary Log' (Entries: 0, Size (KB): 0).

This Web page is an exact duplicate of the *i*.LON 100 Configuration plug-in interface. Configure the Alarm Notifier as described in the *i*.LON 100 User's Guide: *Configuring the i.LON 100 Applications Using the i.LON 100 Configuration Plug-in*.

Latching, Acknowledging, and Clearing Alarms

In some situations, you may want an alarm to stay active even when the condition that caused the alarm has been returned to normal. One example of

this is shown above, with the **Alarm Notifier – Emergency** alarm requiring a manual clear via the Web page.

If a piece of hardware fails, causing an alarm that causes a backup piece of hardware to be activated, the alarm should stay active until manually cleared.

The *i.LON 100* server provides three ways of implementing this behavior:

- Set the **Clear required** checkbox on the **Inputs** tab of the **Alarm Notifier configuration dialog** tab. This method is demonstrated in the example above. Setting this checkbox causes the **Alarm Notifier** to maintain the alarm state until the alarm is cleared using the **Alarm Summary** Web page (see the help for these Web pages for more information). The condition that caused the alarm should be removed or the alarm will just immediately be regenerated. To clear the alarm, a user must have access to the **Alarm Summary** Web page. You can grant this access as described in Chapter 8, *i.LON 100 User Web Page Security*.
- Set the **Acknowledgement required** checkbox on the **Inputs** tab of the **Alarm Notifier configuration dialog** tab. Setting this checkbox causes the Alarm Notifier to maintain the alarm state until the alarm has been Acknowledged using the **Alarm Summary** Web page. An alarm can be acknowledged even while the alarm condition is still true; however, the Alarm Notifier will not terminate the alarm until the condition has been removed.
- Use the **nviAgLatchEnable** network variable on the **Alarm Generator Functional Block**. When this SNVT_switch network variable is set to On (100.0 1), alarms generated by the **Alarm Generator** will persist even when the condition that caused the alarm is no longer true. Set this network variable to Off (0.0 0) to have the **Alarm Generator** return to normal functionality (*i.e.* it will send an alarm only when the conditions on the **Criteria** tab are met).

If you use more than one of these features, you must clear each of them before the alarm will clear. For example, if you set the **nviAgLatchEnable** network variable on an **Alarm Generator** functional block, and send the SNVT_alarm output to an **Alarm Notifier** functional block that has the **Clear Requires** option set, you must set the **nviAgLatchEnable** value to Off (0.0 0) to stop the alarm generator from generating the alarm, and you must also clear the alarm using the **Alarm Summary** Web page to cause the alarm notifier to stop reporting an active alarm condition.

Using the Alarming Web Pages

The *i.LON 100* Web pages include several pages used to monitor, view, acknowledge, and clear alarms. The following Alarming Web pages are available:

Alarm Summary	Allows you to see all active alarms, and to acknowledge and clear alarms. By default, this page lists all alarms. You can filter the alarms shown by Alarm Notifier application, data point, and time of alarm.
Alarm History	Allows you to display a log of active and cleared alarms. You can filter the alarms shown by functional block, data point, and time of alarm.

The Alarm Summary Web Page

To open the **Alarm Summary** Web page, hover your mouse cursor over **View** and select **Alarm Summary**. The following Web page opens:

Submit
Reset

Show:
<All Logs>
<All points>
Poll 5s
Last Poll: 2004-06-18 08:30:48
Manual Read
Log Size: 50KB
Percent Full: 0%
Free disk space: 14210KB

Alarm Summary

Clock may have been manually adjusted.

Select time interval
First log entry 2000-01-01 00:00:00 Last log entry 2038-01-01 00:00:00
Year Month Day Hour Min Sec Year Month Day Hour Min Sec
Start time 2000 Jan 01 00 00 00 End time 2038 Jan 01 00 00 00
Get Range
Go to interval 1 of 1
Viewing interval 1

Alarm time	Location	Point	Pri	Grp	Src	Value	Unit	Description	Comment
------------	----------	-------	-----	-----	-----	-------	------	-------------	---------

This web page allows you to acknowledge and clear active alarms. By default, this page will show all current alarms. An alarm is removed from the Summary log when it is cleared either manually or automatically. If the **Acknowledgement Required** field was checked on the **Inputs** tab of the configuration page for the specified **Alarm Notifier** application instance, the alarm must be manually acknowledged before the **Alarm Notifier** will stop reporting an alarm condition (even if the reason for the condition has been fixed). If the **Clear Required** field was checked on the **Inputs** tab of the configuration page for the specified **Alarm Notifier** functional block, the alarm must be manually cleared using this Web page. To acknowledge an alarm, check the **ACK** checkbox corresponding to the alarm to be acknowledged and click **Submit**. To clear an alarm, check the **CLR** checkbox corresponding to the alarm to be cleared and click **Submit**.

Some alarms require a clear as well as an acknowledge. Typically, the building supervisor will acknowledge the alarm using this web page once the maintenance company has been notified, and the maintenance company will clear the alarm once the problem has been fixed.

You can filter which alarms are shown using the following fields:

Show

Choose a single **Alarm Log** to show alarms from a single **Alarm Notifier** functional block or **<All Logs>**. Choose a single data point to show alarms for a single point or **<All Points>**.

Start Time/End Time

Choose a range of time to view alarms from. The start time is inclusive and the end time is exclusive; i.e. if you set a **Start Time** of 10:00 AM and an **End Time** of 1:00 PM, you will get all alarms from 10:00:00 AM to 12:59:59 PM.

Once you have narrowed the range of alarms, click **Get Range** to show the filtered set of alarms. If the unacknowledged alarms take up too much space, they will be broken down into multiple intervals. You can change which interval is shown by using the slide bar to select an interval and clicking **Get Range**.

For each alarm, the following information is shown:

Alarm Time

The time when the alarm condition changed.

Location	The location property of the data point that triggered the alarm. This is the data point specified on the Inputs tab of the Alarm Notifier application instance.
Point	The name of the data point that triggered the alarm. This is the data point specified on the Inputs tab of the Alarm Notifier application instance.
Pri	The priority of the alarm, as specified on the Inputs tab of the Alarm Notifier application instance.
Grp	The alarm group, as specified on the Inputs tab of the Alarm Notifier application instance.
Src	The LonWorks source address of the data point that triggered the alarm. If this alarm was triggered by a SNVT_alarm or SNVT_alarm_2 type data point, it will be the source address of the device bound to that point.
Value	The value that triggered the alarm condition change. If this alarm was triggered by a SNVT_alarm type data point, this will be the value of the data point that caused the SNVT_alarm or update to be sent (i.e. the value of the Input data point on the Alarm Generator application instance). If this alarm was triggered by a SNVT_alarm_2 point, this value will be the Location of the alarm (i.e. the Location property for Alarm Generator application instance). If the data point is set to a preset value, the preset name will be displayed instead of the actual value.
Unit	The units of the value that triggered the alarm condition change. If this alarm was triggered by a SNVT_alarm or SNVT_alarm_2 type data point, this field will be blank.
Description	The description of the alarm type as set in the Alarms tab of the Alarm Notifier application instance.
ACK	Check this box and click Submit to acknowledge the alarm. If the alarm does not require acknowledgement, this checkbox will not be displayed.
CLR	Check this box and click Submit to clear the alarm. Note that if you clear an alarm when the condition that caused the alarm is still active, the alarm will be regenerated. If the alarm does not require a clear, this checkbox will not be displayed.
Comment	Enter a comment in this field and click Submit to have this comment saved with the alarm. This comment will be added to the alarm log.

When printing this page, use the landscape format.

The Alarm History Web Page

To open the **Alarm History** Web page, hover your mouse cursor over **View** and select **Alarm History**. The following Web page opens:

The screenshot shows the 'Alarm History' web page. At the top, there is a 'Reset' button and a red message: 'Please wait, synchronizing data...'. Below this is a 'Select time interval' section with two rows of date pickers: 'First log entry' and 'Last log entry'. The 'First log entry' row has fields for Year (2000), Month (Jan), Day (01), Hour (00), Min (00), and Sec (00). The 'Last log entry' row has fields for Year (2038), Month (Jan), Day (01), Hour (00), Min (00), and Sec (00). A 'Get Range' button is located below the date pickers. On the left side, there is a 'Show:' section with a dropdown menu set to '<All Logs>', another dropdown set to '<All Points>', and a 'Delete Entire Log' button. Below these are 'Delete Log Interval' and 'Manual Read' buttons. Further down are 'Last Read: ---', 'Log Size: ---', 'Percent Full: ---', and 'Free disk space: 14210KB'. The main content area is a table with the following columns: Alarm time, Log time, Location, Point, Pri, Grp, Src, Value, Unit, Description, ACK/CLR, and Comment. The table is currently empty.

This web page allows you to view the alarm history. By default, this page will show all alarms for a single alarm log. You can determine which alarms are shown using the following fields:

Show

Choose a single **Alarm Log** to show alarms from a single **Alarm Notifier** functional block or **<All Logs>**. Choose a single data point to show alarms for a single point or **<All Points>**.

Start Time/End Time

Choose a range of time to view alarms from. The start time is inclusive and the end time is exclusive; i.e. if you set a **Start Time** of 10:00 AM and an **End Time** of 1:00 PM, you will get all

Once you have narrowed the range of alarms, click **Get Range** to show the filtered set of alarms. Click **Delete Entire Log** to delete the log selected in the **Alarm Log** field. Click **Delete Log Interval** to delete the alarms currently displayed (i.e. filtered by point and time). If the logged alarms take up too much space, they will be broken down into multiple intervals. You can change which interval is shown by using the slide bar to select an interval and clicking **Get Range**.

For each alarm, the following information is shown:

Alarm Time

The time when the alarm condition changed.

Log Time

The time when new information was added to the alarm history. This includes the time that the alarm occurred, which is equal to the alarm time, the time when a comment is added to an alarm, and the time when an alarm has been acknowledged or cleared using the **Alarm Summary** Web page. Acknowledging, clearing, and commenting of alarms may also be performed directly via the SOAP/XML interface of the i.LON

	100, and these updates will also be recorded in the history log in the same fashion.
Location	The location property of the data point that triggered the alarm. This is the data point specified on the Inputs tab of the Alarm Notifier application instance.
Point	The name of the data point that triggered the alarm. This is the data point specified on the Inputs tab of the Alarm Notifier application instance.
Pri	The priority of the alarm, as specified on the Inputs tab of the Alarm Notifier application instance.
Grp	The alarm group, as specified on the Inputs tab of the Alarm Notifier application instance.
Src	The LonWorks source address of the data point that triggered the alarm. If this alarm was triggered by a SNVT_alarm or SNVT_alarm_2 type data point, it will be the source address of the device bound to that point.
Value	The value that triggered the alarm condition change. If this alarm was triggered by a SNVT_alarm type data point, this will be the value of the data point that caused the SNVT_alarm or update to be sent (i.e. the value of the Input data point on the Alarm Generator application instance). If this alarm was triggered by a SNVT_alarm_2 point, this value will be the Location of the alarm (i.e. the Location property for the Alarm Generator application instance). If the data point is set to a preset value, the preset name will be displayed instead of the actual value.
Unit	The units of the value that triggered the alarm condition change. If this alarm was triggered by a SNVT_alarm or SNVT_alarm_2 type data point, this field will be blank.
Description	The description of the alarm type as set in the Alarms tab of the Alarm Notifier application instance.
ACK/CLR	Indicates one of the following: <i>MANUAL_ACK</i> - The alarm was manually acknowledged. <i>MANUAL_CLEAR</i> - The alarm was manually cleared. <i>AUTO_CLEAR</i> - The alarm was cleared when the condition that caused the alarm was removed - no manual clear was required. <i>NACK</i> - The alarm status before it has been acknowledged or cleared - the initial occurrence of the alarm.

Comment

Displays any comments added to the alarm.
Comments may be added on the **Alarm Summary**
Web page.

When printing this page, use the landscape format.

4

Logging Data

This chapter describes how to use the *i*.LON 100 Internet Server to save network data in data logs and how to view the data logs.

Data Logging Overview

The *i*LON 100 Internet Server contains 10 **Data Logger** application instances. You can use a data logger to record updates to a group of user-specified data points into a *log file*. The log files for each data logger are stored in the */root/Data* directory of the *i*LON 100 with the file name logX, where X represents the index number assigned to the data logger.

You can create two kinds of data loggers: *historical data loggers*, and *circular data loggers*. A historical data logger stops recording data point updates when its log file becomes full. A circular data logger removes the records for older updates when its log file is full, and new updates occur. A data logger can save either type of log file in an ASCII-text (.csv file extension) or binary (.dat file extension) format.

The data logger can record data in two ways: event driven updates and polling. Each data point can have its own set of parameters that determine if the data will be recorded only at a specified poll rate (a *poll driven update*), only when an update of the data point occurs (an *event driven update*), or at both a poll rate and when the point is updated. For event driven updates you can set the minimum amount of time between updates and the amount of change in the value, using the **Min Delta Time** and **Min Delta Change** properties. These filters do not affect data that is recorded at a specified poll rate. When a poll rate is set for a data point in a data logger, a value will always be written to the log at that rate.

You can also define a threshold level for each data logger. The threshold level represents a percentage. When the data logger's log file consumes this percentage of the memory space allocated to it, the data logger provides notification that it is time to upload the log and clear out some of the data. The data logger makes this notification by updating the data logger's alarm data point (called NVL_nvoDIAlarm[X], where X represents the index number assigned to the data logger) to the status AL_ALM_CONDITION. This feature is useful when working with historical data loggers, which are disabled when they become full.

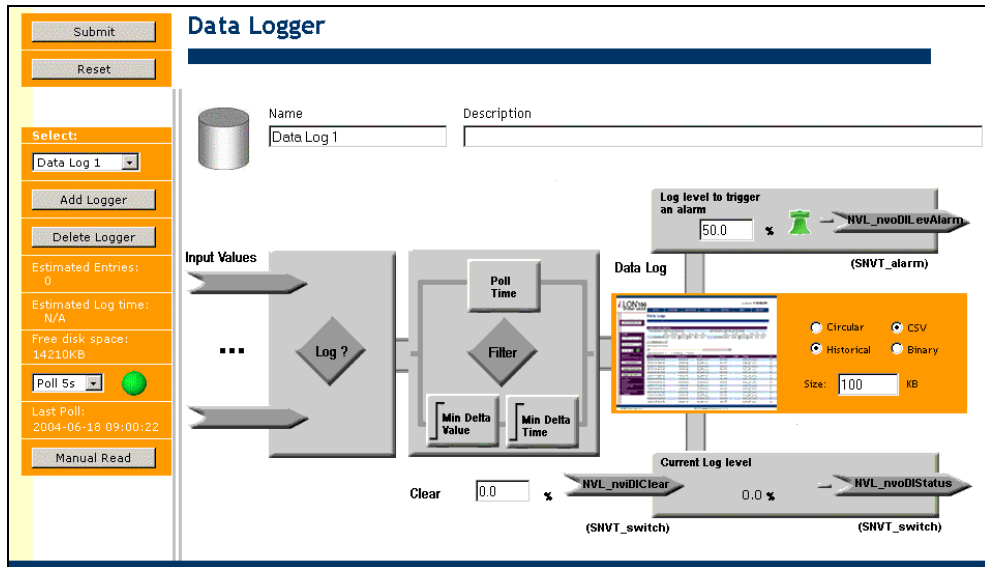
You can access the data in a log file by viewing it in the data logging Web page, by manually opening the log file, or by using a SOAP function. You can clear data from a log by sending an update to the data point NVL_nviDIClear[X], where X represents the index number of the Data Logger to be affected, or by using a SOAP function.

Each log can be up to 1024 KB, and the *i*LON 100 can log up to 10 MB of data. You can view the amount of free disk space using the **System Info** Web page.

Creating a Data Logger

To create a data logger using the *i*LON 100 Web page, follow these steps:

1. Open Internet Explorer 6 or later, point it to the URL of the *i*LON 100, and log in to the *i*LON 100.
2. On the *i*LON 100 Web page, hover your mouse cursor over **Configure** and select **Data Logger** from the drop-down menu. The **Data Logger Configuration** Web page appears, as shown in the following figure:



3. Click the **Add Logger** button to create a new data logger.

4. Enter the following information for the new **Data Logger**:

Name The name of the **Data Logger** application instance.

Description An optional description of the **Data Logger** application instance.

Log Level to Trigger an Alarm When the data log reaches the specified percent capacity, the nvoDILevAlarm output network variable on the associated Data Logger functional block will be set to the AL_ALM_CONDITION status.

Circular/Historical A data log can be either circular or historical. In a historical data log, when the data logger is full, new data will not be logged. In a circular data log, when the data logger is full, the oldest data will be deleted.

CSV/Binary A data log can be stored as either a binary file or a comma separated value file. A binary file can only be read by the View Data Logger Web page. A comma separated value file is a text file that can be read by any application that can read text files, and can be imported into spreadsheet applications such as Microsoft Excel. The name of the log file is log<n>.dat (for binary log files) or log<n>.csv (for comma separated value log files), where <n> is the index number of the Data Logger functional block. Changing the log file format deletes any existing log file.

Size The size in KB, of this data log. Each log can be up to 1024 KB, and the *i*LON 100 can log up to 10 MB of data. You can view the

Clear

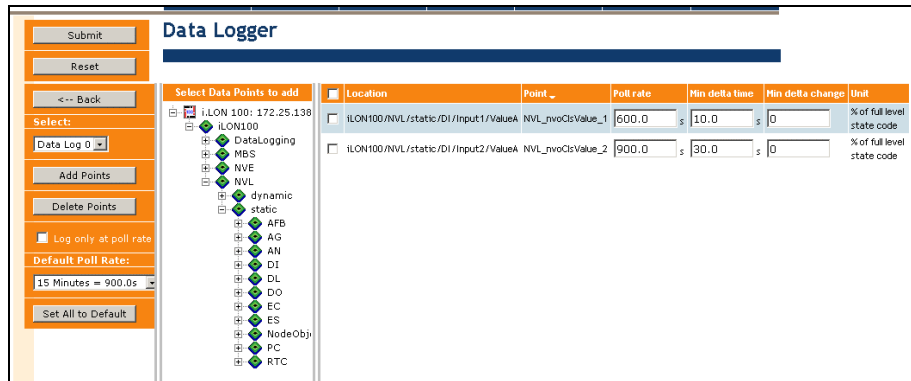
amount of free disk space using the System Info Web page.

This field is used to clear the data log. Enter the percentage of the log you wish to clear and click **Submit** to clear out some or all of the data log. This clears the percentage of the total log. For example, if your data log is only 30% full, entering 60.0 will clear your entire log; if your data log is 90% full, entering 60.0 will leave the log 30% full. Entering 100.0 in this field and clicking **Submit** erases all logged data.

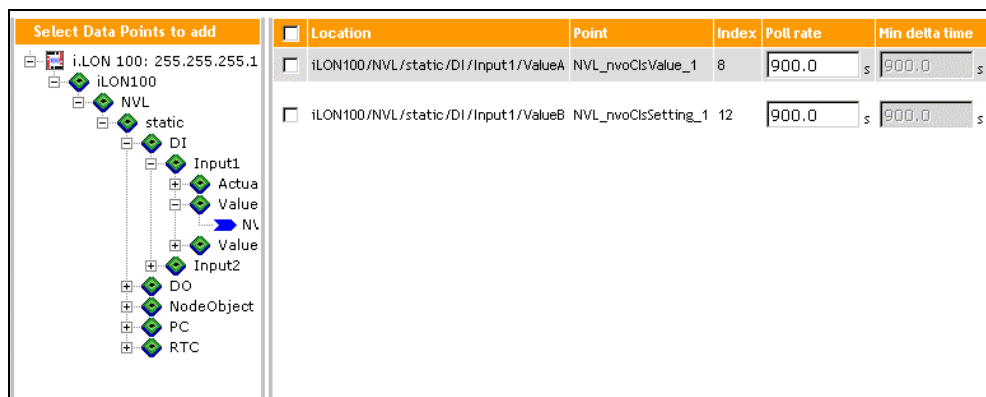
Adding Data Points to a Data Logger

Once you have created a data logger, you must add one or more points to be logged to it. To add data points to a **Data Logger**, follow these steps:

1. On the **Data Logger** Web page, click the **Log?** or **Filter** diamond. The following Web page opens:



2. To add a data point to be logged, browse to the data point to be logged on the left side of this Web page and click the **Add Points** button. You can select multiple points by holding down the <Ctrl> key; selecting a branch of the tree automatically selects all data points contained by that branch. The points will be added to the list of points, as shown in the following figure:



3. Set **Log Only At Poll Rate** on the left sidebar to prevent event driven updates from being written to the log. When this option is set, the Web page sets all **Min Delta Time** values to the corresponding poll rates and all **Min Delta**

Change values to zero and will not allow these options to be modified. This option is set by default.

4. For each point, set the following properties:

Poll Rate How often the data is logged, in seconds. If this value is set to 0, the data will only be logged when it changes in value. You can set a default poll rate for all data points on this Web page from the **Default Poll Rate** drop-down list on the far left side of the Web page. Click **Set All To Default** to have the poll rates for all data points to be logged set to the default value.

Min Delta Change Determines how big of a change needs to be made from the previous logging in order for the point to be logged again. Set **Min Delta Change** to 0 to disable logging on value changes. For structure type data points, set the **Min Delta Change** to any non-zero value to cause any change in value to be logged. This option will be disabled if the **Log Only At Poll Rate** option is set in the left sidebar.

Min Delta Time Determines the minimum amount of time that must pass for a point to be logged again. The data point will not be logged again within this time even if the **Min Delta Change** threshold is passed. This option will be disabled if the **Log Only At Poll Rate** option is set in the left sidebar.

This Web page also displays the following read only information for each point to be logged:

Location The location field of the data point to be logged.

Point The functional name of the data point to be logged.

Unit The units associated with the data point to be logged. This information is derived from the data point's type.

To remove points from a Data Logger, check the box next to the points to be removed and click **Delete Points**.

Extracting Data Logs

You can extract the data logs from the *i.LON 100* server via FTP, SOAP requests, or by having them emailed by the alarm notifier. See the *i.LON 100 Internet Server Programmer's Reference* for information on using SOAP requests. To extract a data log using FTP, follow these steps:

1. Open an FTP client such as Internet Explorer 6.
2. Point the FTP client to the IP address of the *i.LON 100* server (note, FTP access must be enabled on the **Security** Web page as described in the *i.LON*

100 User's Guide: Installing, Connecting, and Configuring the i.LON 100. You will see the i.LON 100 server directory structure.

- Open the **data** folder. You will see a log file for each data logger that you have created. The log files are named log<n>.dat (binary file) or log<n>.csv (semicolon separated value file) <n> is the index of the functional block to which the log applies. The file type (binary or CSV) is determined by the **Format** property on the **Data Logging** tab of the i.LON 100 Configuration Plug-in.
- Copy this log file to your computer using any method available to your FTP client.

Viewing Data Logs

You can view data logs using the i.LON 100 **View Data Log** Web page. To open this Web page, hover your mouse cursor over the **View** menu and select **Data Logs**. The following Web page appears:

Show

Choose the **Data Log** from which to show data. Choose a single data point to show data for a single point or <All Points>.

Start Time/End Time

Choose a range of time to view data from. The start time is inclusive and the end time is exclusive; i.e. if you set a **Start Time** of 10:00 AM and an **End Time** of 1:00 PM, you will get all

Once you have narrowed the range of data to be displayed, click **Get Range** to show the filtered set of data. If the logged data takes up too much space, it will be broken down into multiple intervals. You can change which interval is shown by using the slide bar to select an interval and clicking **Get Range**. Click **Delete Entire Log** to delete the currently selected data log. Click **Delete Log Interval** to delete the currently selected range of data; the Web page only shows the first 60 entries in the range, but the entire range will be deleted. For each log entry, the following information is displayed:

Time

The time at which the data was logged.

Location

The Location property of the logged data point.

Point

The name of the logged data point.

Value	The logged value of the data point.
Unit	The unit associated with the type of the logged data point.
Status	The status of the data point at the time it was logged.
Src	The LonWorks source address of the logged data point.

When printing this page, use the landscape format.

5

Scheduling

This chapter describes how to use the *i.LON 100* Internet Server to create, view, and manage schedules.

Scheduling Overview

The *i*LON 100 contains three types of functional blocks used for scheduling applications—the **Scheduler**, **Calendar**, and **Real Time Clock** functional blocks. The **Scheduler** functional block allows you to define *daily schedules* that each specify the *schedule events* for a 24 hour day. Each daily schedule is either a *day-based* schedule or a *date-based* schedule.

Day-based schedules occur every week on a specified day; each scheduler can contain up to seven day-based schedules (*i.e.* one for each day of the week); you can assign a single day-based schedule to multiple days of the week (*i.e.* you could define one day-based schedule for all weekdays and one for weekends). You can create a *weekly schedule* by assigning a day-based schedule to each day of the week.

Date-based schedules occur on a set of dates defined by an *exception*; each scheduler can contain up to two date-based schedules. Exceptions are ranges of dates defined on the **Calendar** functional block; these can be ranges like “January 12th to February 2nd” or “Every other Monday” or “the third Monday of every month”. The exceptions defined on the calendar can be used by any number of schedulers.

The **Real Time Clock** functional block maintains the current date and time that is used by the **Scheduler** and **Calendar** functional blocks, and can also provide date and time data to other devices. The *i*LON 100 includes one **Real Time Clock** and one **Calendar** functional block and 40 **Scheduler** functional blocks. These functional blocks are described in the following sections.

Creating a Schedule

A typical schedule consists of a weekly schedule that defines the daily schedule to be used for each day of the week, and up to two exception schedules that allow alternate daily schedules to become active on specified dates. Before creating a schedule, you should already have created a LonWorks network that controls the functions you wish to schedule. The steps involved in creating a schedule are:

Step 1: Plan Out Your Schedule – Write out your desired schedule. For example, “On weekday mornings at 6:00 AM the heating system should be started and the thermostat set to 65 degrees. At 8:00 AM, the thermostat should be set to 70 degrees.” etc. See *Planning Out Your Schedule* for more information.

Step 2: Add a Scheduler Application Instance– Use the **Event Scheduler** Web page to create a new Scheduler application instance.

Step 3: Define Outputs – Using the schedule written in step one, decide what data points need to be written by the scheduler.

Step 4: Define Presets – Use the **Data Point Presets** screen of the **Scheduler** application instance to define presets for each of the output network variables defined on the **Data Points** screen. See *Error! Reference source not found.* for more information.

Step 5: Create Daily Schedules– Use the **Scheduler** application instance to define a schedule for each day of the week. Each daily schedule consists of zero or more *events*. An *event* consists of a time and a preset value (*i.e.* 06:00:Warmup). When an event occurs, the data points specified on the **Data Points** screen will be set to the values associated with the preset on the **Data Points Presets** screen.

Each day of the week can have a different schedule. See ***Error! Reference source not found.*** for more information.

Step 6: Create Exceptions – Use the **Calendar** application instance to define exceptions: date ranges during which the weekly schedule will be overridden. These exceptions can be specific ranges of dates (i.e. “March 27th-April 6th, 2002”) or dates in a certain pattern (i.e. “The second Tuesday of every month” or “Every other Saturday”). For each of these exception schedules, use the **Scheduler** functional block to define the schedule that will be used on that day. See ***Error! Reference source not found.*** for more information.

Optional: Create One-time Overrides – You can use the **Scheduler** application instance to create a daily schedule that will override the schedule for a specified range of dates.

Planning Out Your Schedule

Before implementing a schedule using the **Scheduler** and **Calendar** functional blocks, you should write out the schedule so you have a full understanding of all the schedules and events you will need to create.

Example of Planning Out a Schedule

The following is an example of a simple schedule that might be created to schedule HVAC and lighting controls in a retail store:

On weekdays, the heat should be set to 60 degrees at 7:00 AM (Warmup), 70 degrees at 9:00 AM (Open), and turned off at 7:00 PM (Closed).

On weekdays, the lighting should be turned on at 9:00 AM (Open) and turned off at 7:00 PM (Closed).

On weekends, the heating and lighting should remain off.

On the first Sunday of each month, for inventory, the heat should be set to 65 degrees at 9:00 AM (OpenInventory) and turned off at 6:00 PM (Closed).

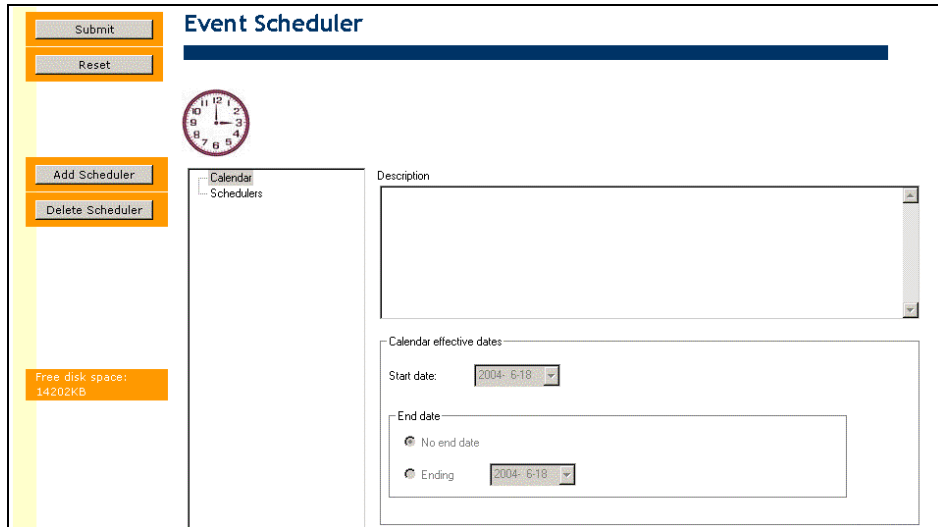
On the first Sunday of each month, for inventory, the lighting should be turned on at 9:00 AM (OpenInventory) and turned off at 6:00 PM (Closed).

Every year you will have a winter vacation. In 2003-2004, you would like it to last from December 23rd to January 3rd, but you would like to be able to change the dates of the vacation every year.

Creating a Scheduler Application Instance

To add a **Scheduler** application instance to the *i.LON 100*, follow these steps:

1. Hover your mouse cursor over the **Configure** menu and select **Event Scheduler**. The following Web page opens:



2. Click **Add Scheduler** to add a new **Scheduler** application instance.
3. The interface for configuring the **Scheduler** and **Calendar** application instances is identical to the interfaces for configuring the **Scheduler and Calendar** application instances using the *iLON 100 Configuration Plug-in*. See the *iLON 100 User's Guide: Configuring the iLON 100 Applications Using the iLON 100 Configuration Plug-in* for more information

6

Using Digital Inputs and Digital Outputs

This chapter describes how to use the *i*.LON 100 Internet Server's two digital inputs and two digital outputs.

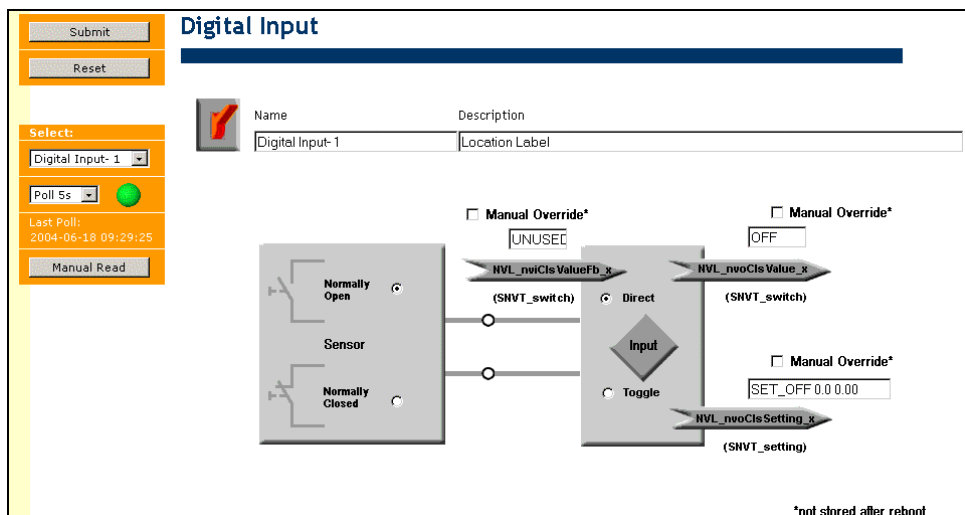
Digital Input Overview

The *i.LON 100* device includes two digital inputs. These can be used to connect the *i.LON 100* device to a digital device that does not include a LONWORKS interface such as a switch or pushbutton. The *i.LON 100* device contains two Digital Input application instances.

Using a Digital Input

To use digital input, follow these steps:

1. Choose one of the two digital inputs. Connect one side of the input to ground and one side to a 12V current source (such as the one provided by +12V < 20mA output – see the *i.LON 100 User's Guide: Installing, Connecting, and Configuring the i.LON 100*) that runs through a switch or dry-contact relay. When no current is received (*i.e.* the relay is open), there is no voltage difference, so the digital input will register On; when current is received, there will be a 12V difference and the digital input will register OFF.
2. On the *i.LON 100* Web pages, hover your mouse cursor over **Configure** and select **Digital Input**. The following Web page opens:



3. Select **Digital Input - 1** or **Digital Input - 2** from the **Select** drop-down menu on the left side of the Web page to determine which **Digital Input** you will use.
4. Enter the following information:

Normally Open/Normally Closed

Select **Normally Open** to have the **Digital Input** application send an Off value when the hardware input is open. Select **Normally Closed** to have the **Digital Input** application send an Off value when the hardware input is closed.

Direct/Toggle

Select **Direct** to have the **Digital Input** application send the current state of the hardware input. Select **Toggle** to have the Digital Input application toggle its output each time the hardware input changes from

Manual Override

Off to On.

You can set manual override values for each of the three data points used by a **Digital Input** application instance (NVL_nviClsValueFb_x, NVL_nvoClsValue_x, and NVL_nvoClsSetting_x). Enable the override value for a data point by checking the **Manual Override** option. For NVL_nviClsValueFb_x, the i.LON 100 will use the override value as an input and ignore other updates to this data point; for the two output data points, the i.LON 100 will send the override values regardless of the state of the hardware input. The override value must be properly formatted (or use a defined preset) for the data point type (SNVT_setting for NVL_nvoClsSetting_x, SNVT_switch for the other two data points).

After changing any of these options, it is recommended that you resynchronize the i.LON 100 to the LNS database at the next opportunity.

5. Click **Submit** to save the settings.

Digital Output Overview

The i.LON 100 device includes two dry-contact relay outputs. These can be used to connect the i.LON 100 device to a digital device that does not include a LONWORKS interface such as a drive contactor or alarm bell. The i.LON 100 device contains two Digital Output application instances.

Using a Digital Output

To use a digital output, follow these steps:

1. Connect the one of the two relay outputs to a digital input.
2. On the i.LON 100 Web pages, hover your mouse cursor over **Configure** and select **Digital Output**. The following Web page opens:

Digital Output

Submit
Reset

Select:
Digital Output-1

Poll 5s

Last Poll:
2004-06-18 09:38:58

Manual Read

Name	Description
Digital Output-1	Location Label

Manual Override*
OFF

Manual Override*
OFF

*not stored after reboot

3. Select **Digital Output - 1** or **Digital Output - 2** from the **Select** drop-down menu on the left side of the Web page to determine which **Digital Output** you will use.

4. Enter the following information:

Normally Open/Normally Closed

Select **Normally Open** to have the **Digital Output** application switch the hardware output off when the Off value is received. Select **Normally Closed** to have the **Digital Output** application switch the hardware input on when an Off value is received.

Manual Override

You can set manual override values for each of the two data points used by a **Digital Output** application instance (NVL_nviClaValue_x, and NVL_nvoClaValueFb_x). Enable the override value for a data point by checking the Manual Override option. For NVL_nviClaValue_x, the i.LON 100 will use the override value as an input and ignore other updates to this data point; for NVL_nvoClaValueFb_x, the i.LON 100 will send the override value regardless of the state of the hardware input. The override value must be properly formatted (or use a defined preset) for the data point type (SNVT_switch).

After changing any of these options, it is recommended that you resynchronize the i.LON 100 to the LNS database at the next opportunity.

5. Click **Submit** to save the settings.

7

Using Pulse Counter Inputs

This chapter describes how to use the *i*.LON 100 server's Pulse Counter to measure electrical energy, volume, rate or flow, or power.

Pulse Counter Overview

The *i*LON 100 device includes two pulse counter inputs. These can be used to connect the *i*LON 100 to a power meter or other device with a pulse output that does not include a LONWORKS interface. The *i*LON 100 device contains 2 Pulse Counter application instances. If you want to change the type of data that is sent by the Pulse Counter application instances, you must use the *i*LON 100 Configuration Plug-in to change the output data point types; these types cannot be changed using the **Pulse Counter** Web page.

Using the Pulse Counter

To use pulse counter, follow these steps:

1. On the *i*LON 100 Web pages, hover your mouse cursor over **Configure** and select **Pulse Counter**. The following Web page opens:

2. Select **Pulse Counter - 1** or **Pulse Counter - 2** from the **Select** drop-down menu on the left side of the Web page to determine which **Pulse Counter** you will use.
3. Enter the following information:

Name	The name of this Pulse Counter application instance.
Description	An optional description of this Pulse Counter application instance.
Multiplier/Divisor	These fields allow you to input a multiplier and divisor to convert the units per pulse. For example, if your power meter sends 1 pulse per 10 watt hours, set the Multiplier to 10 and the Divisor to 1 to translate the data to watts/hour.
Time Interval	This field appears below PcValue next to the NVL_nvoPcValueDif_x data point shape. Set the interval that will be used as the divisor for the NVL_nvoPcValueDif_x value. This value

also determines how often the output data point values are updated.

Trigger

Click this button to automatically start a new time interval. This is used to resynchronize the pulse counter.

Manual Override

You can set manual override values for the NVL_nvoPcValue_x and NVL_nvoPcValueDif_x output data points used by a **Pulse Counter** application instance. Enable the override value for a data point by checking the **Manual Override** option. When this option is checked, the i.LON 100 will send the override value regardless of hardware input. The override value must be properly formatted (or use a defined preset) for the data point type

Store

To set the NVL_nvoPcValue_x data point to a specified starting value, set this option, enter the value in the **Manual Override** field, and click **Submit**. The NVL_nvoPcValue_x data point will be set to this value and the Store option will be automatically unchecked.

After changing any of these options, it is recommended that you resynchronize the i.LON 100 to the LNS database at the next opportunity.

8

Creating a Web Page

This chapter describes how you can use the *i*LON 100 to serve custom Web pages that you create in addition to the configuration Web pages described in this document. You can use these Web pages to serve data point values, allowing your user to read and write data point values via the custom Web pages.

Introduction

The *i.LON 100* implements an XML Web service interface that uses SOAP messaging to provide Internet access to the data and applications in the *i.LON 100*. This interface is covered in complete detail in the *i.LON 100 Programmer's Guide*. This chapter describes how to create *i.LON 100* Web pages that can monitor and control data points using SOAP messages.

Previous versions of the *i.LON 100* used Web tags to serve data point values on user Web pages. This technique is still available for backwards compatibility and is described in *Appendix A*. Web tag support may be removed in a future release of the *i.LON 100*, so creating Web interfaces using the techniques described in this chapter is recommended.

This chapter contains 10 tutorials that will guide you through the process of creating a custom *i.LON 100* Web page that serves data point values using SOAP messages. A background in Web programming and JavaScript will help you understanding this example, but it is not a requirement to be able to modify the example for a custom application.

The first four tutorials describe how you can modify this tutorial Web page for use in a custom Web page. The last six tutorials describe the various parts of the JavaScript SOAP library provided in the SOAP Tutorial Web page. You can use this Web page as a template for new projects. You will only need to modify a few isolated sections of JavaScript and HTML for a custom application.

User Web Page Locations

You can upload user Web pages to the *i.LON 100* using any FTP client. The *i.LON 100* provides a space for custom Web pages in the following location:

```
/root/Web/user/index.html
```

You can open the Web page in this location by clicking the **Custom** menu on any *i.LON 100* Web page.

The tutorials in this chapter use the SOAP tutorial Web page, which ships on the *i.LON 100* in the following location:

```
/root/Web/user/examples/SoapTutorial.html
```

You can place new user Web pages in the examples directory or in a new directory that you create in `/root/Web/user` such as:

```
/root/Web/user/MyCompany/MyWebPage.html
```

Tip: Since Web pages outside of the **forms** directory do not contain dynamic content; these files may be statically compressed to improve first load performance over a slow Internet connection. To compress these files for the *i.LON*, use the open source tool called Gnu Zip, from <http://www.gzip.org>. Once you finish development of your Web page, simply gzip the file and place the compressed version on the flash disk next to the uncompressed version like this:

```
/root/Web/user/MyCompany/MyWebPage.html
```

```
/root/Web/user/MyCompany/MyWebPage.html.gz
```

Communicating with the DataServer

You will use are **DataServerRead** and **DataServerWrite** SOAP methods to read and write the values of data points on an *i.LON 100*. A single SOAP message

may contain an array of data points of any bus type: NVL, NVE, NVC, MBS, etc. Though this tutorial covers only the **DataServerRead** and **DataServerWrite** SOAP methods, you can apply the concepts described in this chapter to other SOAP methods covered in the *i.LON 100 Programmer's Guide*.

The Microsoft Web Service Behavior

Microsoft provides a free tool for Web developers to add SOAP to IE 6.0+ Web pages. This tool is called the Microsoft DHTML Web Service Behavior. You can add this behavior to any Web page by including the **webservice.htc** file on the Web server and referencing the file in the **<BODY>** tag of a Web page as shown in the following code:

```
<BODY ID="ilonService"  
STYLE="behavior:url(../../WSDL/webservice.htc);">
```

A modified version of the **webservice.htc** file, which includes a few bug fixes, is available in the following location on the i.LON 100:
`/root/Web/WSDL/webservice.htc`

Beginning Tutorials

The following 4 tutorials describe how you can modify **SoapTutorial.html** to create a custom Web page. These tutorials will show how to modify this tutorial Web page to read and write data points, how to change the poll rate, and how to customize the display. These tutorials are designed for users with limited HTML and JavaScript experience.

Tutorial 1: Reading Data Points

This tutorial covers the modification of the Web page included on the i.LON 100 at:

`/root/Web/user/examples/SoapTutorial.html`

To view this Web page in Internet Explorer, use the following address:

`http://my-ilon-ipaddress/user/examples/SoapTutorial.html`

A copy of this page is installed on your computer by the i.LON 100 installation in the following location:

`$LONWORKS$\iLON100\Images\iLON100
1.10\Web\user\Examples\SoapTutorial.html`

Before editing this file, create a backup copy that you can refer to later. You can open this file using a text editor or an HTML editor such as MacroMedia DreamWeaver; if you are using an HTML editor, be sure that the editor will not make additional modifications to the html code.

Step 1: Create a list of data points

The SOAP Tutorial uses a JavaScript array called `g_dpList` to store a list of data points whose values will be polled via SOAP when the page loads. Locate the definition of the JavaScript array in the SOAP Tutorial source code:

```
var g_dpList = new Array();
```

A JavaScript object represents each data point in the array. To read a data point, you need to supply its name, which is stored in the property called `UCPTpointName`. By default, the SOAP Tutorial will poll the values of the two

digital output network variables and their feedback variables. To create a data point object, add it to the array and set the **UCPTpointName** property, only the following two lines of code are required:

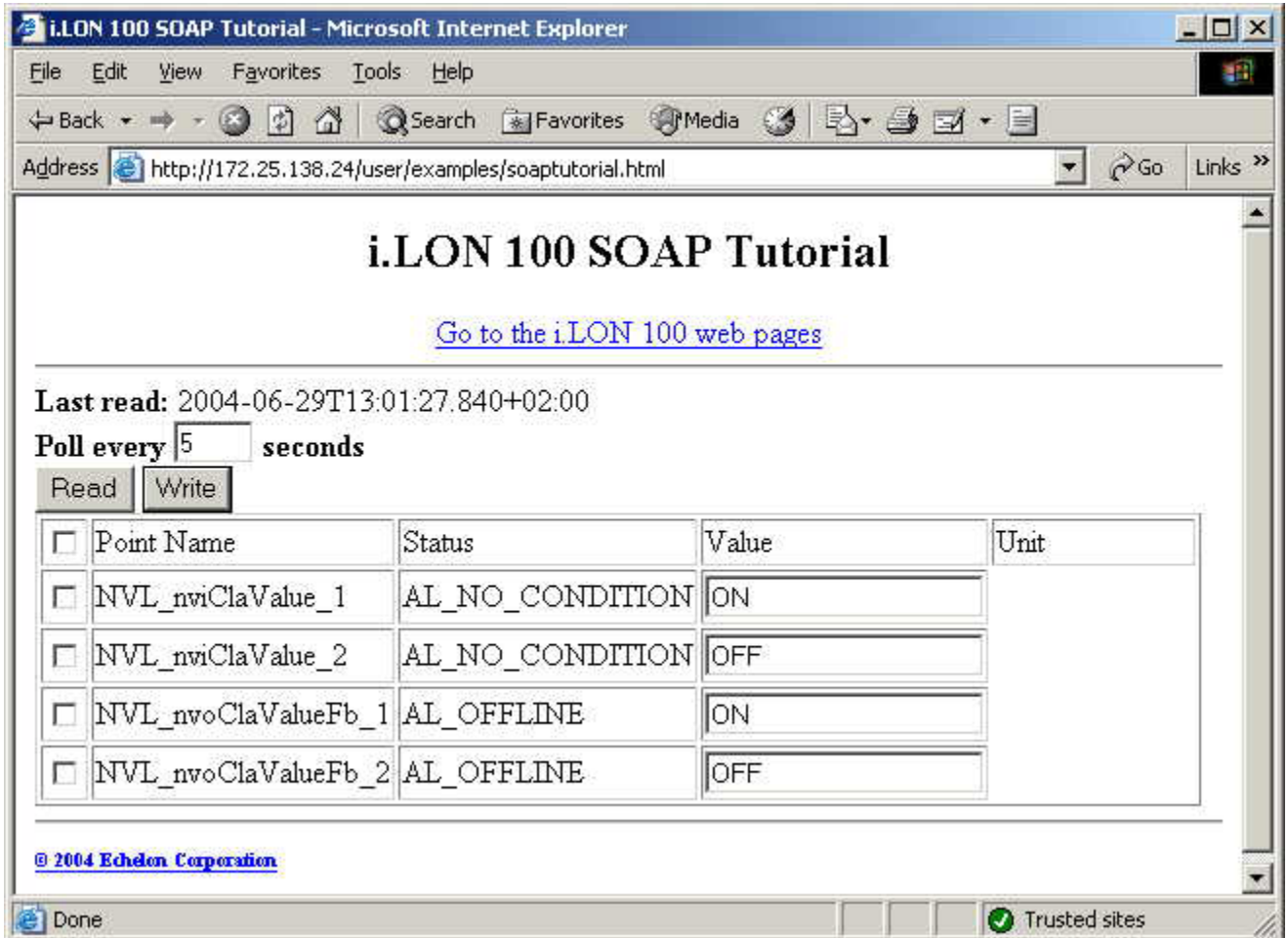
```
g_dpList[0] = new Object();  
g_dpList[0].UCPTpointName = "NVL_nviClaValue_1";
```

Add these two lines (substituting the appropriate data point name) for each data point you would like to view. Increment the array index by 1 for each data point you wish to add, as shown in the following code:

```
g_dpList[1] = new Object();  
g_dpList[1].UCPTpointName = "NVL_myTemperature";
```

Step 2: View the Web Page

Once you have finished editing the data point array, FTP the file to `/root/Web/user/examples/SoapTutorial.html`, overwriting the old version. Refresh the Web page to see the data points that you have defined. The page should appear similar to the following figure:



Tutorial 2: Customizing the Display

This tutorial focuses on customizing the HTML page and linking the objects to the data point code.

Step 1: Referencing HTML Objects

Scroll to the bottom of the SoapTutorial.html source code to find the HTML code for the text boxes, buttons, and data point table. In JavaScript, there are several ways to reference these HTML objects. One way to do this is to use the **id** property of the object. It is important for all objects in a single HTML page to have unique **id** properties to ensure that this type of reference returns a single unique object. To reference the body of the table where the values of the data point array will be written, first find the HTML which defines the table:

```
<table border="1">
  <thead>
    <tr>
      <td><input id="allBox" type="checkbox"
onclick="checkAll();"></td>
      <td>Point Name</td>
      <td width="100">Status</td>
      <td>Value</td>
      <td width="100">Unit</td>
    </tr>
  </thead>
  <tbody id="dpTable" onkeydown="checkOne();">
  </tbody>
</table>
```

The code between the **<thead>** elements defines the headers that are displayed across the top of the table. The data point values will be written in the **<tbody>** section of the table, and has the **id dpTable**. Next, define a global variable in JavaScript that will be used for a reference the body of the table:

Step 2: Initializing the Display

In the SOAP Tutorial a function called **initPage** starts the SOAP Web service and initializes the display when the **onload** event fires. After the Microsoft Web Service Behavior (see *The Microsoft Web Service Behavior*, earlier in this chapter) has been started, **initPage** calls another function that you can modify called **initUI**. This function initializes the global reference to each HTML object used by the browser. To get a reference to an HTML object using the **id** property, use the **getElementById** function of the built-in document object, which takes as an argument the unique **id** of the HTML object:

```
function initUI() {
  // build the references to some important DHTML objects
  g_tableObj = document.getElementById("dpTable");
  g_tsObj = document.getElementById("lastPollField");
  g_pollRateObj = document.getElementById("pollRateField");
```

The **initUI** function also iterates through the data point array used in the first tutorial and adds a row to the table for each element in the array. Your customized **initUI** function should set the initial state of the graphical objects you create.

Step 3: Creating Custom HTML Objects

Now that these objects can be referenced in the JavaScript code, you can create HTML by hand or using one of the many HTML design packages on the market

such as Macromedia Dreamweaver, Adobe GoLive, or Microsoft FrontPage. Once you have added graphics, text boxes, tables and other graphical objects to your page, you can add references to those objects as shown above. The following HTML example creates a simple text box that displays a temperature:

```
<input type="text" id="myTemperature" size="5">
```

Next, create a global variable for that text box:

```
var g_tempBoxObj;
```

Finally, add the object reference to the **initUI** function:

```
function initUI() {  
    g_tempBoxObj = document.getElementById("myTemperature");
```

Step 4: Displaying Data Point Values

The response to the DataServerRead SOAP message includes the data point's name and value. You can use this response to write the value of a data point to the text box that you created in step 3. The SOAP Tutorial has a poll task that reads all of the data points in **g_dpList**, and calls a function called **displayValues** every time the response to the poll arrives. That function takes one argument called **dsResponse**, which is a JavaScript object. This object has a property called **AoDP** that is an array in the exact same format as **g_dpList**. Find the **displayValues** function and create a for loop that can iterate through the array in the response object and find a data point to display:

```
for (var a=0; a<dsResponse.AoDP.length; a++) {  
    if (dsResponse.AoDP[a].UCPTpointName == "NVL_myTemperature") {
```

To write the value to the text box, use the **value** property of the text box object. The value of the data point is stored in the **UCPTvalue** property:

```
        g_tempBoxObj.value = dsResponse.AoDP[a].UCPTvalue;  
    }  
}
```

The complete list of properties for the data point object is as follows:

dpObj. UCPTindex [string]	Index of the data point for the complete list for the bus (NVL, NVE, etc.)
dpObj. UCPTpointName [string]	Name of the data point
dpObj. UCPTpointUpdateTime [string]	Time stamp indicating the last time the point received any type of update from the network, from SOAP, or from an application like the Scheduler
dpObj. UCPTvalue [string]	Formatted value of the data point

<code>dpObj.UCPTvalueDef</code> [string, optional]	If the current value of the data point is the same as one of the Preset values for the point, this property will display the Preset value; for example a <code>SNVT_switch</code> value may have a definition of "ON" for "100.0 1"; if no preset matches the current value this property will be undefined for the object
<code>dpObj.UCPTunit</code> [string]	Unit of measurement for the data point
<code>dpObj.UCPTpointStatus</code> [string]	Enumerated value of the current status of the data point; this can be set by the driver for the data point or an application like the Alarm Generator
<code>dpObj.UCPTpriority</code> [string]	Priority of the data point; if a SOAP request does not have higher priority than the data point, no update will be made; this provides a mechanism for user interfaces to adhere to a hierarchy of write privileges

Tutorial 3: Changing the Poll Rate

After every **DataServerRead** SOAP message the value in the poll rate text box determines the number of seconds to wait before the next poll. If the user enters zero, polling stops. If polling fails for some reason, polling stops. You can call **stopPolling** at any time to end the polling loop and clear the current poll. If polling stops, the **Read** button will start polling again.

In this tutorial, you will customize the poll rate to fit your application. First determine how quickly changes to the data must be displayed. For example, temperature values that change every few minutes can be updated every few minutes; the state of a physical switch should be updated with no more than one second of delay. Next, determine if the user will have the ability to change the poll rate or if the script will manage it. Finally, determine if the user will have the ability to stop polling completely.

Step 1: Choosing a Default Poll Rate

Over a 100Mbit LAN connection with very short delays of less than 10ms, it is possible to poll 100 NVL data points with a 1 second poll rate in a Web page like **SoapTutorial.html**. However, such performance may not be necessary if you only need to view the update of a few temperature values every five minutes, and it may not be possible if you are connected to the *i.LON 100* via modem or connected across the world with delays greater than 100ms. To choose a default poll rate that works for your application, you must know the properties of your network, and that means that the users of your Web pages will connect to the *i.LON 100*. To set the default poll rate for the SOAP Tutorial, modify the declaration of **g_pollRate**, the global variable that stores the poll rate:

```
var g_pollRate = 5;
```

Tip: Built-in to every computer is a tool called **ping**, which provides useful information about the connection between your computer and a host on the Internet like an *iLON 100*. For example, you can use this tool to determine the response time from sending the ping command to receiving the ping response. You can run this tool from the Windows command prompt. It displays the round trip time of a packet to a specified destination in milliseconds. To ping an *iLON 100* at 192.168.1.222, enter the following command:

```
C:\>ping 192.168.1.222
```

The ping command will return a response that looks similar to the following:

Pinging 192.168.1.222 with 32 bytes of data:

```
Reply from 192.168.1.222: bytes=32 time=161ms TTL=126
Reply from 192.168.1.222: bytes=32 time=160ms TTL=126
Reply from 192.168.1.222: bytes=32 time=160ms TTL=126
Reply from 192.168.1.222: bytes=32 time=160ms TTL=126
```

Ping statistics for 192.168.1.222:

```
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 160ms, Maximum = 161ms, Average = 160ms
```

Step 2: Reading User Input

In `SoapTutorial.html`, the reference to the text box is called `g_pollRateObj` and the you can read the string representation of the data entered into the field from the `value` property. The `getPollRate` function performs a few checks on this string and returns an integer to the SOAP poll task to set the timer for the next read. To adjust the poll rate for your application, modify the `getPollRate` function and be sure that it returns a valid positive integer. The following section describes how to use a select box that lists a set of predefined poll rates. To hard code the poll rate, change the function to return the global variable for the poll rate:

```
function getPollRate() {
    return g_pollRate;
}
```

To change this to a select box with a default of 5 seconds, you can replace the text box with the following code:

```
<select id="pollRateField">
  <option value=0>Disable polling</option>
  <option value=1>Poll 1s</option>
  <option value=5 selected>Poll 5s</option>
  <option value=10>Poll 10s</option>
  <option value=20>Poll 20s</option>
  <option value=30>Poll 30s</option>
  <option value=60>Poll 60s</option>
</select>
```

In this case, the user cannot input an invalid value, they can only select one of the valid choices from the box. Thus, the **getPollRate** function becomes rather simple:

```
function getPollRate() {
    return g_pollRateObj.value;
}
```

Tutorial 4: Writing to Data Points

In the previous tutorials a text box displayed the value of a temperature data point. In this tutorial you will use the same text box to collect input from the user and send it in a **DataServerWrite** SOAP message to the *i*.LON 100. This will allow the user to write a data point by typing into the box for that point, making sure that the check box is selected, and then clicking **Write**.

Step 1: Calling the writePoints Function

The following code defines a button that will call the **writePoints** function:

```
<BUTTON onclick="writePoints();">Write</BUTTON>
```

The **writePoints** function, which is defined in the SOAP Tutorial Web page, calls the **getUserInput** function before sending the **DataServerWrite** SOAP message to the *i*.LON 100. You will modify this function to read the value from the text box.

Step 2: Getting User Input

The **getUserInput** function reads the values in the table and creates an array of data point objects. The following code creates an array with a single data point object to represent **NVL_myTemperature**.

```
function getUserInput() {
    var AoDP = new Array();
    var newDpObject = new Object();
    newDpObject.UCPTpointName = "NVL_myTemperature";
```

To read the value, access the **value** property of the text box. To write the value to the data point, set the **UCPTvalue** property of the data point object and add the data point object to the array.

```
    newDpObject.UCPTvalue = g_tempBoxObj.value;
    AoDP[0] = newDpObject;
```

The **getUserInput** function must return an array of data point objects; use the JavaScript keyword **return** to pass this back to the caller function:

```
    return AoDP;
}
```

Step 3: Handling a Successful Response

The JavaScript SOAP Tutorial has two functions for handling the response of a **DataServerWrite** message—**resetUI** and **displayWriteErrors**. The **resetUI** function is called when no errors are returned by the i.LON 100. In **SoapTutorial.html**, this function clears all of the checkboxes and sets the color of all text boxes to white (the **displayWriteErrors** function, described below, may change the color of a text box). In your application you may want to reset the state of some graphics, clear an input field, enable a control, or some other manipulation of the user interface. In this example, we will set the background of the text box to green if the write was successful.

```
function resetUI() {
    g_tempBoxObj.style.backgroundColor = "green";
}
```

Step 4: Handling Errors in the Response

When there is an error in a **DataServerWrite** request the response will contain a detailed list of errors detected for each data point submitted. If you enter “abc” for a temperature value, the i.LON 100 will return an error message stating that the value is formatted incorrectly. This information is returned in array of data point objects with two special properties to carry error information, **faultcode** and **faultstring**. The **faultcode** property is an integer that an application may use to display different messages for different types of errors. The **faultstring** property contains a string describing the error. The following code checks to see if an error code was returned:

```
function displayWriteErrors(errList) {
    for (var e=0; e<errList.AoDP.length; e++) {
        if (errList.AoDP[e].UCPTpointName ==
"NVL_myTemperature") {

// If there is an error display a message to the user about
// the error and mark the field red. The error code for a
// format error is 11. The following code checks for a
// format error:

if (errList.AoDP[e].faultcode == "11") {
    alert("The value entered for NVL_myTemperature is out of range or invalid.");
else {
    alert("An error occurred while writing to NVL_myTemperature.");
}

// The following code sets the color of the field to red:

g_tempBoxObj.style.backgroundColor = "red";
        }
    }
}
```

Advanced Tutorials

The previous tutorials involved modifying the existing **SoapTutorial.html** Web page. The following tutorials describe how to create a new Web page like

SoapTutorial.html. The advanced tutorial require a basic understanding of JavaScript scripting.

Tutorial 5: Enabling SOAP in the Browser

This tutorial covers the process of attaching the Microsoft Web Service Behavior to an Internet Explorer 6.0+ compatible Web page, and preparing the Web page for SOAP communication.

Step 1: Attach the Microsoft DHTML Web Service Behavior to a Web Page

Add the following code to the top of the Web page, as described in *The Microsoft Web Service Behavior*, earlier in this chapter:

```
<BODY ID="ilonService"
STYLE="behavior:url(../../WSDL/webservice.htc);">
```

Step 2: Define Links to the WSDL File and SOAP Path

In the <HEAD> of the Web page, add a block where you can insert JavaScript code, as shown in the following code:

```
<HEAD>
<SCRIPT type="text/javascript">
<!--
// my JavaScript code goes here

// -->
</HEAD>
```

Alternately, you can add a link to an external JavaScript file where all of the JavaScript code described in the following sections will go, as shown in the following code:

```
<SCRIPT type="text/javascript"
src="../../scripts/examples/SoapTutorial.js"></SCRIPT>
```

Once you have defined the JavaScript code area of the Web page, add the following global variables to the beginning of the code:

g_soapPath	The location on the <i>i.LON 100</i> where the Web browser sends SOAP requests. This path is the same as the location of the V1.0 WSDL file.
g_wsdlPath	The location of the V1.1 WSDL file. The WSDL file defines the SOAP interface of the <i>i.LON 100</i> , and it is used by the Microsoft Web Service Behavior to build the request messages and handle the responses.

The following code adds these variables to an *i.LON 100* Web page:

```
var g_soapPath="../../WSDL/iLON100.WSDL";
var g_wsdlPath="../../WSDL/V1.1/iLON100.WSDL";
```

Step 3: Start the Web Service Behavior

When a web page loads, it fires the **onload** event called which may be acted upon by a script. To define an event handler function for the **onload** event for the entire page, add the following attribute to the <BODY> tag:

```
<BODY ID="ilonService" onload="startService();"
STYLE="behavior:url(../../WSDL/webservice.htc);">
```

Next, define a global variable to store a reference to the Web Service Behavior object once it has been started:

```
var g_ilonService;
```

Last, define the **startService** function in the JavaScript code area.

```
function startService() {
    var success=false;
    // get a reference to the body object which will also serve as
    // the reference to the Web Service Behavior
    g_ilonService=document.getElementById("ilonService");
    try {
        // call the function useService with the WSDL path as the first
        // parameter and a shortcut name for the service
        g_ilonService.useService(g_wsdlPath,"iLON100");
        success=true;
    }
    catch (e) {
        // upon failure, show the text of the error message
        var alertStr = "An error occurred while starting the Microsoft ";
        alertStr += "Web Service Behavior. The error was:\n\n";
        alertStr += e.description;
        alert(alertStr);
    }
    return success;
}
```

Tip: When creating multiple Web pages that send SOAP messages to the iLON 100, the Web pages will start faster if the Web Service Behavior is only started once for all Web pages. The easiest way to do this is to create an HTML frameset and load the **WSDL** and **webservice.htc** files into the top level Web page of the frameset. Each Web page that is loaded into that frameset may then use the Web service object from the top frame. Creating links between JavaScript functions in different frames is beyond the scope of this tutorial, but it is covered in many books on JavaScript and DHTML.

Tutorial 6: Sending Low Level SOAP Messages

This tutorial describes the JavaScript code required to send a SOAP message and receive the response.

Step 1: Add a Button to the Web Page

Once the Web Service Behavior has been started, any script may send SOAP messages. In this tutorial a button defined in the HTML of the Web page will trigger an **onclick** event and call the **readPoints** function. First, define a button inside the <BODY> of the web page:

```
<BUTTON onclick="readPoints();" >Read</BUTTON>
```

Next, define the **readPoints** function in the JavaScript code area:

```
function readPoints() {  
  
}
```

Step 2: Create the Request String

This step uses the **DataServerRead** SOAP method to illustrate the example. For complete details on this method please refer to the *i.LON 100 Programmer's Reference*. The *i.LON 100* has two digital outputs connected physically to relays which may be connected to external hardware. **NVL_nviClaValue_1** and **NVL_nviClaValue_2** are the names of the data points that are inputs to the two **Digital Output** application instances. To read the values of data points using the **DataServerRead** SOAP message, the request must have the following XML structure:

```
<iLONDataServer>  
  <NVL>  
    <UCPTpointName>NVL_nviClaValue_1</UCPTpointName>  
  </NVL>  
  <NVL>  
    <UCPTpointName>NVL_nviClaValue_2</UCPTpointName>  
  </NVL>  
</iLONDataServer>
```

The tabs and carriage returns are not required in the string, so the JavaScript variable can be defined as follows:

```
function readPoints() {  
  var requestStr = "<iLONDataServer><NVL>";  
  requestStr += "<UCPTpointName>NVL_nviClaValue_1</UCPTpointName>";  
  requestStr += "</NVL><NVL>";  
  requestStr += "<UCPTpointName>NVL_nviClaValue_2</UCPTpointName>";  
  requestStr += "</NVL><NVL></iLONDataServer>";
```

Step 3: Send the SOAP Message

Sending SOAP messages using the Microsoft Web Service Behavior always involves a few required steps. You can create a generic function called **soapRequest** that performs these steps for any SOAP message. This function will take the following arguments:

method [string]	The name of the SOAP method to send to the <i>iLON</i> 100, such as DataServerRead .
request [string]	The XML data used as the input to the SOAP method.
callback [function]	Called when the response to the SOAP message returns.

```
function soapRequest(method, request, callback) {
    var co = g_ilonService.createCallOptions();
    co.funcName = method;
    co.endpoint = g_soapPath;
    co.params = new Array();
    co.params.Data = encodeXml(request);
    var id = g_ilonService.iLON100.callService(callback, co);
    return id;
}
```

To make this function work you must convert the XML to an acceptable format before sending the SOAP message. Since SOAP is actually based on XML, the payload of the message may not contain XML if the input parameter is defined as a string. Thus, all `<` and `&` characters must use the encoded versions: **<** (`<`), **>** (`>`), and **&** (`&`). The **encodeXml** function used above takes care of this conversion for us, and thus we can pass the string created in the previous step directly to **soapRequest**, as shown in the following code:

```
function readPoints() {
    var requestStr = "<iLONDataServer><NVL>";
    requestStr += "    <UCPTpointName>NVL_nviClaValue_1</UCPTpointName>";
    requestStr += "</NVL><NVL>";
    requestStr += "<UCPTpointName>NVL_nviClaValue_2</UCPTpointName>";
    requestStr += "</NVL><NVL></iLONDataServer>";

    soapRequest("DataServerRead", requestStr, handleReadPoints);
}
```

This completes the **readPoints** function. When the button is clicked, the SOAP message will be sent. Upon receiving the response, the Web Service Behavior will call the **handleReadPoints** callback function, which we will define in the next section.

Step 4: Handle the Response

The callback function called by the Web Service Behavior takes the response object as an argument. The properties of this object are described in detail in the Microsoft documentation for the Web Service Behavior. The following properties

of the response object are described for this tutorial (the object in the callback function will be called **res**):

res [object]	Response from the Microsoft Web Service Behavior; for more information, see the Microsoft documentation.
res.error [boolean]	Indicates error in the response.
res.errorDetail.code [string]	Indicates an error from the client (i.e. Web browser), and the Server (i.e. meaning <i>i.LON 100</i>).
res.errorDetail.string [string]	Description of the error
res.errorDetail.detail [MSXML object]	Optional error object property containing details of a DataServerWrite failure.
res.value [string]	Raw XML of the response from the <i>i.LON 100</i> .
res.SOAPHeader [object]	Includes all properties in the SOAP header.
res.SOAPHeader.UCPTtimeStamp [Date object]	Formatted timestamp of the response from the <i>i.LON 100</i> .
res.SOAPHeader.UCPTtimeStamp.raw [string]	Unformatted timestamp of the response from the <i>i.LON 100</i> .
res.raw [MSXML object]	Raw SOAP message

The first step in the response handler is to check error property:

```
function handleReadPoints(res) {
    if (!res.error) {
        // handle a successful response
        alert("Your response message is:\n\n" + res.value);
    }
    else {
        // handle errors
        var alertStr = "Error writing to data points.";
        alertStr += "\n res.errorDetail.code: " + res.errorDetail.code;
        alertStr += "\n res.errorDetail.string: " + res.errorDetail.string;
        alert(alertStr);
    }
}
```

Tutorial 7: Converting XML to Native JavaScript Objects

In the previous tutorials code was created for sending and receiving raw XML strings in the SOAP messages. In this section some simple techniques will be described for converting these XML structures into native JavaScript objects so that your applications can build on top of a clean object interface rather than depending on a series of complex XML parsing steps.

Step 1: Creating a Request Object

To send a request for a group of data points, the following XML structure is used:

```

<iLONDataServer>
  <NVL>
    <UCPTpointName>NVL_nviClaValue_1</UCPTpointName>
  </NVL>
  <NVL>
    <UCPTpointName>NVL_nviClaValue_2</UCPTpointName>
  </NVL>
</iLONDataServer>

```

Working with this XML structure as a string or as a true XML object can add many extra lines of code to a simple project. To avoid this, a function has been added to this tutorial to convert the above XML structure to JavaScript objects. To create a data point object and set the **UCPTpointName** property, only the following lines of code are required:

```

var dpObj = new Object();
dpObj.UCPTpointName = "NVL_nviClaValue_1";

```

To create an entire array of these data points, use the following code:

```

var g_dpList = new Array();
g_dpList[0] = new Object();
g_dpList[0].UCPTpointName = "NVL_nviClaValue_1";
g_dpList[1] = new Object();
g_dpList[1].UCPTpointName = "NVL_nviClaValue_2";

```

In the previous tutorials, global properties were added to the **DataServerRead** message. You can step up one more level and create an object to hold the entire request message:

```

function readPoints() {
  var dsRequest = new Object();

```

Add a property called **AoDP** to include the “Array of Data Points”, and set that property equal to the global array of data points defined above:

```

dsRequest.AoDP = g_dpList;

```

Finally, call the conversion function **obj2Xml** on the **dsRequest** object before sending it to the **soapRequest** message. You can perform a check on your request array to make sure that there is something to send:

```

if (dsRequest.AoDP.length>0) {
  soapRequest("DataServerRead", obj2Xml(dsRequest), handleReadPoints);
}
else {
  alert("ERROR: There are no data points in the data point array.");
}
}

```

Step 2: Converting the Response XML to Objects

When the SOAP response is received, the `xml2Obj` function that has been added to this tutorial can be called to convert the response to XML. To get an object that corresponds to the response, use the following code:

```
function handleReadPoints(res) {
    if (!res.error) {
        var dsResponse = xml2Obj(res.value);
    }
}
```

The response object can be passed to a Web page specific display function:

```
displayValues(dsResponse);
```

A display function such as this can either iterate through the array of data points returned from the SOAP request as shown below

```
function displayValues(dsResponse) {
    for (var a=0; a<dsResponse.AoDP.length; a++) {
```

or get a single data point from the list:

```
var myDpObject = AoDP[0]; // this gets the first data point
in the array
```

In tutorial 2 the following properties were described:

```
dpObj.UCPTindex
dpObj.UCPTpointName
dpObj.UCPTpointUpdateTime
dpObj.UCPTvalue
dpObj.UCPTvalueDef
dpObj.UCPTunit
dpObj.UCPTpointStatus
dpObj.UCPTpriority
```

Any of these properties can be assigned to a variable in JavaScript by accessing the property. Here are examples for the single object and the for loop defined above:

```
var currentTemperature = myDpObject.UCPTvalue;

var measurementUnit = AoDP[a].UCPTunit;
```

Tutorial 8: Writing to Data Points

A similar method to that shown in Tutorial 7 can be used to write a value to a data point. In this tutorial a button will be created that writes a value from a text box to a data point.

Step 1: Add a Button to the Web Page

First, create a button whose **onclick** event will trigger a JavaScript function:

```
<BUTTON onclick="writePoint();" >Write</BUTTON>
```

Next, define the **writePoint** function in the JavaScript code area:

```
function writePoint() {  
  
}
```

Step 2: Add a Text Box to the Web Page

Define a text box by using the **<input>** HTML element:

```
<input type="text" id="valueField" size="20">
```

To access the value entered by a user, get the HTML object using the **id** property, then access the **value** property of the text box.

```
function writePoint() {  
var textBoxObject = document.getElementById("valueField");  
var currentValue = textBoxObject.value;
```

Step 3: Create the Request Object

As with the read function, create a request object that has the **AoDP** property to hold an array of data points.

```
function writePoint() {  
    var textBoxObject = document.getElementById("valueField");  
    var currentValue = textBoxObject.value;  
    var dsRequest = new Object();  
    dsRequest.AoDP = new Array();
```

Next create a new element in the array using the zero index, and will set the **UCPTpointName** and **UCPTvalue** properties of this data point object:

```
    dsRequest.AoDP[0] = new Object();  
    dsRequest.AoDP[0].UCPTpointName = "NVL_nviClavalue_1";  
    dsRequest.AoDP[0].UCPTvalue = currentValue;
```

Step 4: Send the Message

Use **obj2Xml** in combination with **soapRequest** to send the data to the *iLON 100*:

```
soapRequest("DataServerWrite", obj2Xml(dsRequest),  
handleWritePoints);  
}
```


Step 5: Handle the Response

Upon success, the response message is not very interesting. The XML for the response to the above message would be:

```
<iLONDataServer>
  <NVL>
    <UCPTindex>20</UCPTindex>
  </NVL>
</iLONDataServer>
```

However, if the user enters “900.0 1”, the value will be out of range, and the i.LON will not update the data point. The i.LON 100 will respond to this error with a SOAP fault. The **DataServerWrite** message uses an optional property of the fault message called **detail** to display an array of fault codes and fault strings for each error in the request message. The detail property contains a block of XML that appears as follows:

```
<iLONDataServer>
  <AoDP>
    <DP>
      <UCPTindex>20</UCPTindex>
      <UCPTdataPointType>NVL</UCPTdataPointType>
      <UCPTpointName>NVL_nviClaValue_1</UCPTpointName>
      <faultcode>11</faultcode>
      <faultstring>Value cannot be formatted</faultstring>
    </DP>
  </AoDP>
</iLONDataServer>
```

To handle these errors, use the functions from the previous tutorial to change this XML into a JavaScript array:

```
function handleWritePoint(res) {
  if (!res.error) {
    alert("success");
  }
  else {
    // if the fault message uses the detail property,
    // parse it and display errors for each data point
    if ((res.errorDetail.detail != null) &&
        (typeof(res.errorDetail.detail)=="object")) {

      var errList = xml2Obj(decodeXml(res.errorDetail.detail.xml));
      var errString = "The error for " + errList.AoDP[0].UCPTpointName;
      errString += " is " + errList.AoDP[0].faultstring;
      alert(errString);
    }
  }
}
```

Tutorial 9: Using Presets – the UCPTvalueDef Element

An important feature of the DataServer is the use of Presets that convert formatted values such as “100.0 1” for a SNVT_switch into a more user friendly string constant such as **ON** (see Chapter 2 of the *The i.LON 100 User's Guide: Configuring the i.LON 100 Applications Using the i.LON 100 Configuration Plug-in* for details). These values can be configured using the *i.LON 100 Configuration plug-in* or the *i.LON 100 Web pages*, and they are stored in a property of the data point called **UCPTvalueDef**. When reading data points, the **UCPTvalueDef** element is an optional parameter of the response. If the value of the data point matches one of the Presets exactly, the corresponding string constant will be sent in the **UCPTvalueDef** element of the **DataServerRead** response. When writing to a data point, the **UCPTvalueDef** element may be used to write a Preset value to the data point. In fact, for **DataServerWrite**, the **UCPTvalue** element is optional, and the **UCPTvalueDef** element may be used to send either the Preset string or the formatted value such as “100.0 1”. This tutorial shows how to modify the read and write examples from the previous sections to use Presets.

Step 1: Adding a Text Box to the Web Page

Only one text box is required to display the Preset value or formatted value of the data point. This same text box may be used by the write function to write either the Preset value or the formatted value to the data point. The following code creates a text box:

```
<input type="text" id="valueField" size="20">
```

Step 2: Reading and Displaying Data Point Presets

The only change to the code shown in Tutorial 1 required to display preset values is in the response handler. Getting the response object and convert it to a JavaScript object:

```
function handleReadPoints(res) {
    if (!res.error) {
        var dsResponse = xml2Obj(res.value);
```

Assume that the request contains only a single point. To access that point, get the first element from the response array:

```
var dpObject = dsResponse.AoDP[0];
```

When extracting the value from the data point object, determine if the optional **UCPTvalueDef** property has been sent in this message:

```
var textBoxObject = document.getElementById("valueField");
if (dpObject.UCPTvalueDef != null) {
    textBoxObject.value = dpObject.UCPTvalueDef;
}
```

```
else {
    textBoxObject.value = dpObject.UCPTvalue;
}
```

Step 3: Writing to Data Points Using Presets

To write to data points using Presets, only a single line of code from Tutorial 4 must be changed in the **writePoint** function:

```
dsRequest.AoDP[0].UCPTvalueDef = currentValue;
```

By writing to the data point using the **UCPTvalueDef** property, the text box will now accept any of the Preset values for the data point such as **ON** and **OFF** or any properly formatted value such as “75.0 1”.

Tutorial 10: Polling Data Points in JavaScript

This tutorial builds on top of the previous sections by adding a poll task to the **readPoints** function. The **DataServerRead** SOAP method can be used to read a large list of data points. To maximize the performance of this method, a time stamp may be added to the request indicating that only data points whose value has changed since that time should be sent back in the response. This allows large number of data points to be observed in a single Web page with a relatively small amount of data polled in the background. To get the timestamp for the next message the poll task uses the timestamp from the SOAP header of the previous message.

Step 1: Start Reading Data Points

To start the poll task, call the **readPoints** function. This sends the first **DataServerRead** SOAP message, and the **handleReadPoints** callback function will be called when the response arrives:

```
<BUTTON onclick="readPoints();" >Read</BUTTON>
```

Step 2: Read the Time Stamp of the Response

When a response arrives, the time stamp may be read from the header of the SOAP message. The header is available as a property of the response object called **res.SOAPHeader**. We will store the raw string value of the time stamp in the **g_lastPollTs** global variable. This raw format string will be used by the poll task in the next message to filter the response.

```
function handleReadPoints(res) {
    if (!res.error) {
        // first store the time stamp from the message
        g_lastPollTs = res.SOAPHeader.UCPTtimeStamp.raw;
    }
}
```

For your reference, the **SOAPHeader** object has several properties:

UCPTtimestamp [Date object]	JavaScript Date object containing the date and time of the last message; this object has a special property called raw which stores the original string from the <i>i.LON 100</i> .
UCPTuniqueId [hexadecimal integer]	Neuron ID of the <i>i.LON 100</i> device; can be used to uniquely identify <i>i.LON 100</i> devices
UCPTipAddress [string]	Current IP address of the interface over which the SOAP message was sent: Ethernet for LAN, modem for PPP
UCPTport [integer]	TCP/IP port used by the interface over which the SOAP message was sent; by default the <i>i.LON 100</i> Web server uses port 80

Step 3: Start the Timer for the Next Read

First, get the poll rate from the user interface. Tutorial 3 describes a function used to retrieve this value in seconds, so write the poll rate to **g_pollRate**:

```
g_pollRate = getPollRate();
```

Next start the timer if the poll rate is not zero:

```
if (g_pollRate > 0) {
```

To start a timer in JavaScript, use the built-in **setTimeout** function of the **window** object. This function takes as arguments a JavaScript expression, such as a function call, and the number of milliseconds to wait before evaluating that expression. This function returns an **id** for a **timer** object that may be used to clear the timeout before it expires. Store the **timer** object **id** in the **g_pollTimer** global variable.

```
g_pollTimer = window.setTimeout("readPoints();", g_pollRate * 1000);
```

Step 4: Copy the Time Stamp to the Next Message

The **UCPTstart** property is used to optimize the response to the **DataServerRead** message; it must be included at the top level of the **dsRequest** object:

```
if (g_lastPollTs != "") {
    dsRequest.UCPTstart = g_lastPollTs;
}
```

In the XML representation appears as follows:

```

<iLONDataServer>
  <UCPTstart>2004-06-29T14:54:16.520+02:00</UCPTstart>
  <NVL>
    <UCPTpointName>NVL_nviClaValue_1</UCPTpointName>
  </NVL>
  <NVL>
    <UCPTpointName>NVL_nviClaValue_2</UCPTpointName>
  </NVL>
</iLONDataServer>

```

It is important to note that only data points which have been updated since the **UCPTstart** time will be returned in the response. Thus, you cannot guarantee that the length of the request array will be equal to the length of the response array.

Step 5: Stopping the Poll Task

A special utility function has been added to SoapTutorial.html to stop the poll task. This function uses built-in function of the **window** object called **clearTimeout** to stop the timer. This function takes as a parameter only the **id** of the **timer** object that was returned by **setTimeout** in step 3. If that timer has already expired when **clearTimeout** is called, an exception may occur; a **try / catch** statement can be used to trap the exception. For this tutorial this exception can be ignored, as it indicates only that no timer is currently active. To prevent further polls set **g_pollRate** to zero.

```

function stopPolling() {
  try {
    window.clearTimeout(g_pollTimer);
  }
  catch (e) {
    // if an exception occurs, it means that the timer was already cleared
  }
  g_pollRate = 0;
}

```

Using Other Tools To Build *i*.LON 100 Web Pages

Several tools are available to speed up and simplify the process of creating *i*.LON 100 Web pages. These tools include:

- Macromedia Dreamweaver and other HTML development environments. These tools provide many features designed to facilitate the rapid development of Web pages. For example, Macromedia Dreamweaver allows you to create and save code snippets; these snippets can be easily dropped into an existing page. You can use this feature to create generic code for reading or writing data points and then customize it for your application by changing just a few lines. See the documentation for your HTML development tool for more information.
- Webgen. Webgen is an unsupported Echelon tool designed to facilitate quick development of *i*.LON 100 Web pages. Use this tool to quickly put together a basic Web structure with pages that automatically update and animate data point values. Once an initial set of Web pages is created with Webgen, the

pages can be modified using standard web tools. The Webgen tool is available for download on the Echelon Web site at www.echelon.com/ilon.

9

***i*.LON 100 User Web Page Security**

This chapter contains information on restricting viewing and modification of user created *i*.LON 100 Web pages.

Overview of *i*.LON 100 User Web Page Security

You can restrict access to files under the *i*.LON 100 server's `/root/Web` directory to provide Web page security. You may secure the access by user name/password, source IP address, or location of the resource (URL).

You will define Web page security using the *i*.LON Web Server Parameters utility. This utility is included with the *i*.LON 100 software and is accessible from the *i*.LON 100 program folder (**START > PROGRAMS > Echelon *i*.LON 100 > *i*.LON 100 Web Server Security and Parameters**). The *i*.LON Web Server Parameters utility creates a `WebParams.dat` file that you must transfer via FTP to the *i*.LON 100's root directory. (`/root/WebParams.dat`)

The *i*.LON 100 device parses the `WebParams.dat` file on startup to establish Web page restrictions. The `WebParams.dat` file is stored as plain text with no encryption or password protection. This means that *i*.LON 100 security is protected from inspection by FTP security (user name and password) only. Be sure to set proper user names and passwords for FTP access to prevent the `WebParams.dat` file from being viewed (see ***Error! Reference source not found.*** in Chapter ***Error! Reference source not found.***). Also, be sure to secure the computer that you use to create the `WebParams.dat` file.

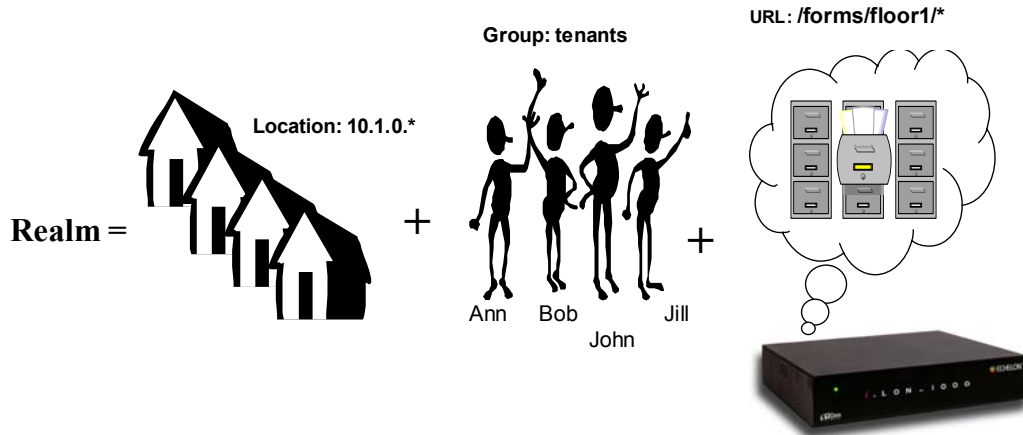
The *i*.LON 100 server's factory default `WebParams.dat` file allows access to all files found under `/root/Web` from any location by any user. To modify the existing *i*.LON 100 Web security you have to create a new (or edit the existing) `WebParams.dat` file. After modifying the `WebParams.data` file, transfer the updated file using an FTP application to the *i*.LON 100, and then reboot the *i*.LON 100 for the new security settings to take effect.

To change the security settings on an *i*.LON 100 server, follow these steps:

1. Download the existing `WebParams.dat` file from the `/root` directory of the *i*.LON 100 using an FTP application.
2. Start the *i*.LON Web Server Parameters application, and open `WebParams.dat` (File > Open command).
3. Make the required security changes (see below) and save the `WebParams.dat` file using the File > Save command.
4. Upload the `WebParams.dat` file to the *i*.LON 100's `/root` directory using an FTP application.
5. Reboot the *i*.LON 100 to activate the security changes.

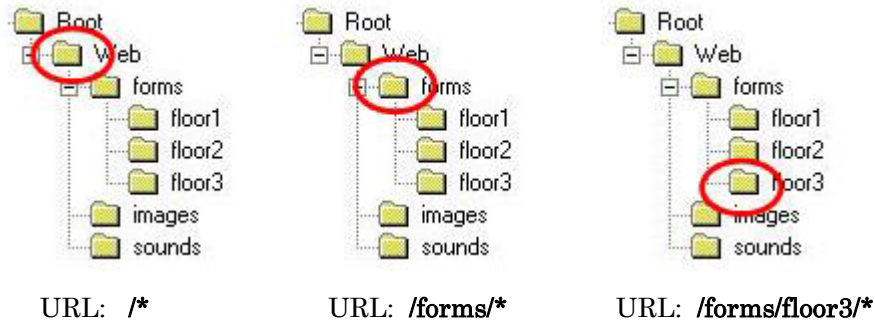
Setting Access Restrictions

Security realms are used to define *i*.LON 100 server access restrictions. A *realm* is defined to be the combination of URL (folder in *i*.LON 100 server), group (users group name), and location (IP address range from where the URL may be accessed). In other words, a realm defines which files (URL) may be accessed by which group of users (group) and from which IP addresses (location).



An iLON 100 Server Security Realm

URLs are defined with the assumption that you are starting from the root of the Web site and not the iLON 100 server. For example, to restrict access to `http://building10/forms/floor3/` the URL must be defined as `"/forms/floor3/*"`. The wildcard is required in order to place this setting across the entire directory. To restrict access to the whole site you need to use the URL `"/*"`. The following figure shows examples of URLs (**the leading "/" is required syntax**):



Users and Groups

Each person who will be given access (*i.e.* a user name and password) to the iLON 100 server is called a *user*. Users are organized into groups. Each user can be in exactly one group, and all users in a given group will have identical access. A group must contain one or more users. A group can contain a maximum of 16 users; this limit is not enforced by the Web Server Parameter application; if you add more than 16 users to a group, the iLON will ignore all users after the 16th (*i.e.* they will not be considered part of the group). If each user must have different access rights, you must define a group for each user.

In order to define a group you must first define a list of users and passwords, for example,

```
Ann : boxcar
Bob : trumpet
John : foxtrot
Jill : mustang
superuser : sfs43fs6f
```

Users are then grouped together based on the iLON 100 Web folders that they are going to access. For instance, if Ann, Bob, Jill and John live in the same building, you could group them by floor. Ann, Bob, and Jill have apartments on

the second floor, Bob also happens to have a workshop on the first floor. Finally, John has an apartment on the third floor. The property management company maintains the Web site. Their Web master has the access name superuser. The following table shows which users are to have access to which folders:

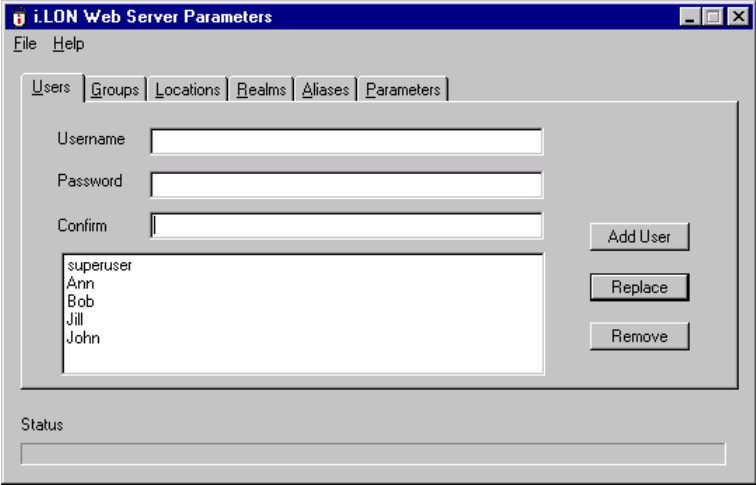
Example iLON 100 Web Page Security Chart

	floor 1	floor2	floor3
Ann		x	
Bob	x	x	
Jill		x	
John			x
superuser	x	x	x

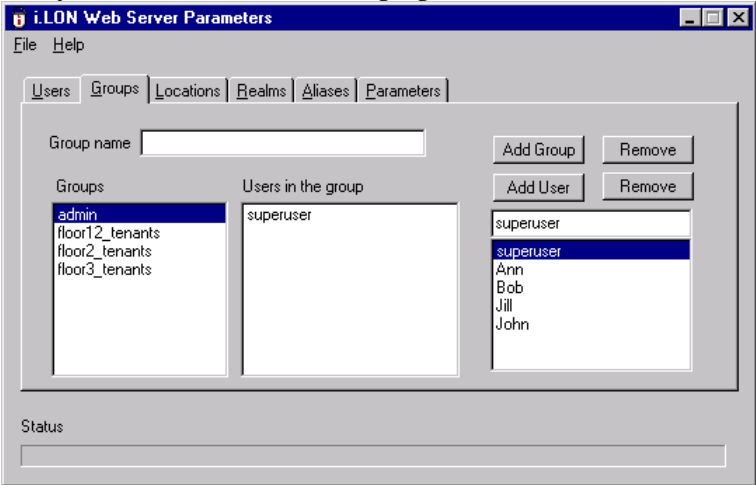
The iLON security mechanism allows each user to be a member of one group only. Thus, the 4 groups will need to be created. One each for access to floors 1 & 2 (Bob), floor 2 (Ann, Jill), floor 3 (John), and all floors (superuser):

To set up the users and groups described above, follow these steps:

1. Setup usernames and passwords from the **Users** tab of the *iLON Web Server Parameters* utility as shown in the following figure:



2. Once all the user names and passwords have been entered, create the necessary groups using the **Groups** tab of the *iLON Web Server Parameters* utility as shown in the following figure:



- Finally, add users to specific groups by selecting the group and then clicking the **Add User** button for each user you want to add to the group.

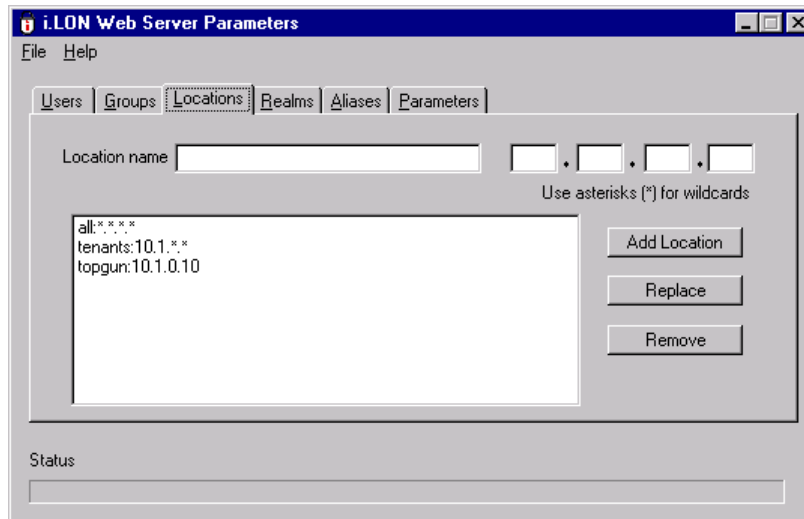
NOTE: If you create 16 or more groups of users, you may need to change some of the parameters as described in *Parameters*, later in this chapter.

Locations

Locations are defined as ranges of IP addresses from which a particular group of users can access a particular folder. “*” is used as a wildcard. Examples:

Location name	IP address range	Comments
All	*.*.*.*	Any IP address
Tenants	10.1.0.*	Any host with IP in the range 10.1.0.1 – 10.1.0.254 Note that 10.1.0.0 is a network address and 10.1.0.255 is a broadcast address, hence they are not included
Topgun	10.1.0.10	IP address of the host used by superuser (property manager) to update Web pages

Use the *i.LON Web Server Parameters* utility’s **Locations** tab to define these locations as shown in the following figure:



If you declare a location “A” that happens to be a subset of another location “B,” it is assumed that “A” is not included in the access rights of users in location “B.”

For example:

```
topgun: 10.1.0.10
tenants: 10.1.*.*
all: *.*.*.*
```

This declaration actually means that `tenants` is the whole range `10.1.*.*` with the exception of `10.1.0.10`. Similarly, `all` excludes `10.1.*.*`.

Realms

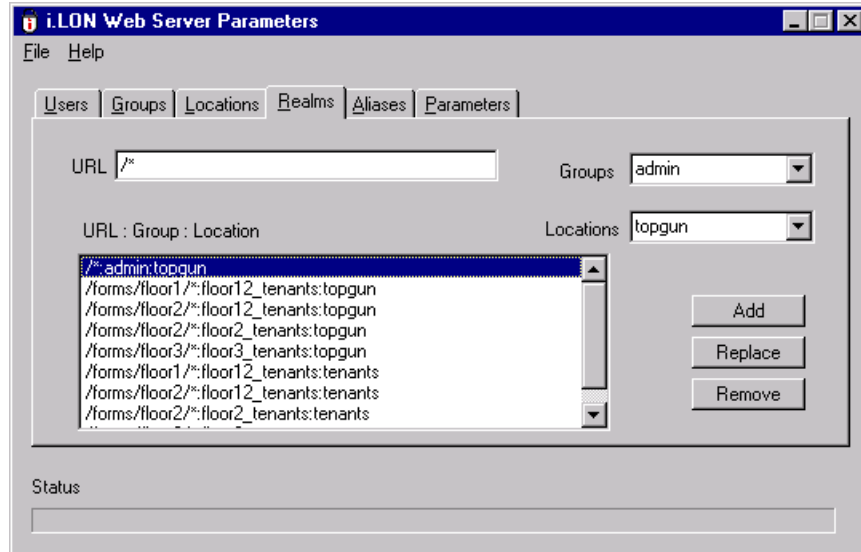
Realms define the folders the various groups and locations are allowed to access. Each realm is in the format `URL:GROUP:LOCATION`, where users from `GROUP` and `LOCATION` are given access to the `URL`. These values can be selected in the **Realms** tab of the *i.LON Web Server Parameters* utility.

For example, design a security setup for an *i*LON 100 Web site that allows users to monitor occupancy information, temperature, and light level on the floor on which they live. This is a three-story building so we have floors 1, 2, and 3, with corresponding Web pages stored in folders under `/forms/`: `/forms/floor1`, `/forms/floor2`, and `/forms/floor3`. There are five users that can access this site: superuser, Ann, Bob, Jill, and John. They belong to groups `tenants_floor12`, `tenants_floor2`, `tenants_floor3`, and `admin` as described above.

Tenants are allowed to access Web pages of their floor only, but can login from any local host;

Local hosts may have any IP address in the network `10.1.0.0 / 24` (*i.e.* `10.1.0.1 – 10.1.0.254`). There is one “superuser” that designs Web pages, and has unlimited access to the Web site; for security reasons he or she will access the site from one host only, with IP address `10.1.0.10`; the Web site should be restricted to any other users.

Based on this description, the **Realms** tab should appear as shown in the following figure:



Aliases

Aliases allow redirecting URLs to other URLs in the web server directory structure. Use this procedure to define aliases for creating cross-references or to define realms for web page security. A realm identifies a URL to the web pages, and a group and location name that has access to those web pages. See *Realms*, earlier in this chapter, for instructions on setting up realms.

The syntax for an alias is: `URL:Path`. The following example redirects a request made with the URL element `/forms/DIRA/Nvpage.html` and converts it to `/secureforms/Nvpage.htm` redirects all URLs ending in a slash to the `index.htm` file in the same path.

`/forms/DIRA/*:/secureforms/*`

Tip: Use the asterisk (*) as a wildcard in both the URL string and alias string. It must occur only once in the URL string and once in the alias string.

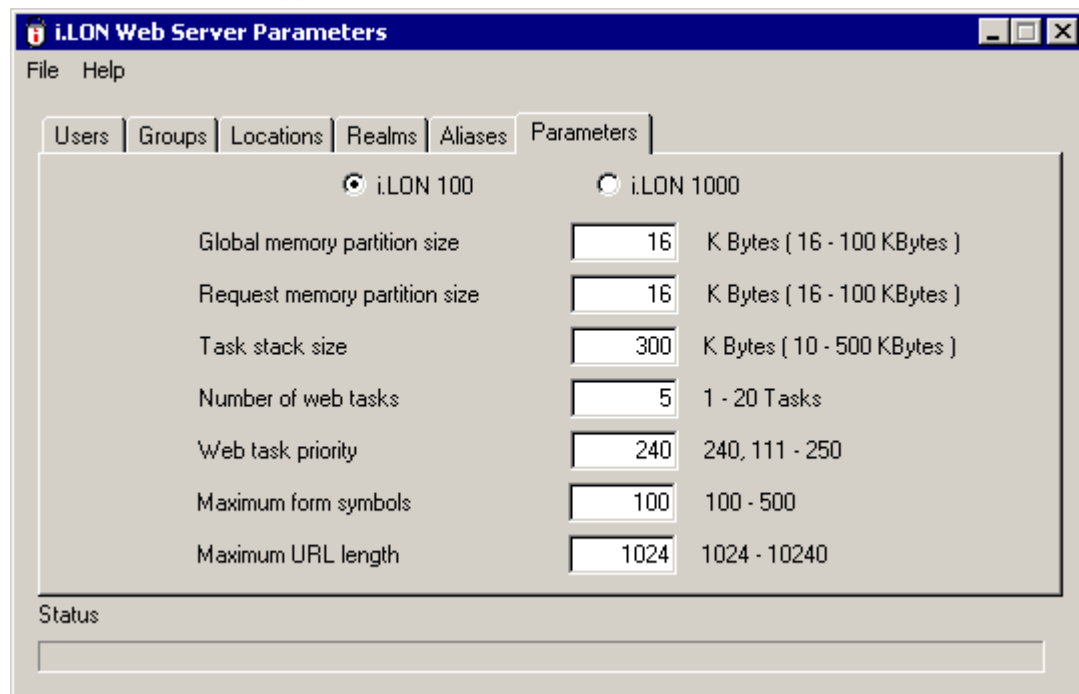
To create an alias:

1. From the *i*LON Web Server Parameters main dialog, select the **Aliases** tab. The Aliases dialog appears.

2. In **URL**, enter the URL that identifies the path to the web pages for which you wish to define an alias.
3. In **Path**, enter the path to which you wish to redirect.
4. Click **Add**.
5. Your new alias definition appears in the window.

Parameters

The **Parameters** tab appears as shown in the following figure:



Click **Help** for more information about these parameters.

Typically, you will not change any of the values in this tab. However, if you create 16 or more groups of users and you have trouble accessing the *i.LON 100* server, the following changes are recommended:

Global memory partition size	80KB
Request memory partition size	80KB
Maximum form symbols	200
Maximum URL length	2048

Sample WebParams.dat file

The following WebParams.dat file was generated according to the scenario discussed above.

```
iLonSecurity 1.2
GlobalMemoryBytes:16384
RequestMemoryBytes:16384
TaskStackBytes:10240
NumTasks:1
TaskPriority:240
MaxSymbols:100
(Users)
```

```
admin:superuser:sfs43fs6t
floor12_tenants:Bob:trumpet
floor2_tenants:Ann:boxcar
floor2_tenants:Jill:mustang
floor3_tenants:John:foxtrot
(Locations)
all:*. *.*.*.*
tenants:10.1.*.*
topgun:10.1.0.10
(Realms)
/*:admin:topgun
/forms/floor1/*:floor12_tenants:topgun
/forms/floor2/*:floor12_tenants:topgun
/forms/floor2/*:floor2_tenants:topgun
/forms/floor3/*:floor3_tenants:topgun
/forms/floor1/*:floor12_tenants:tenants
/forms/floor2/*:floor12_tenants:tenants
/forms/floor2/*:floor2_tenants:tenants
/forms/floor3/*:floor3_tenants:tenants
```

Appendix A

Creating a Web Page Using Web Tags

This appendix demonstrates how to create Web pages that serve Web page values using web tags. The SOAP method described in Chapter 8 is the preferred method. Web tags may not be supported in future releases of the *i.LON 100*.

Overview of Creating *i.LON 100* Web Pages Using Web Tags

The *i.LON 100* server's embedded Web server application and the *i.LON 100* server's embedded data server application work together to serve Web pages that reference network variables to a standard Web browser.

The *i.LON 100* server's data server provides an anchor point to which both input and output network variables can be bound. Additionally, it knows how to pass the current value of network variables to the Web server, and how to accept data from the Web server to propagate to its output network variables.

Prior to serving a Web page to a browser, the Web server parses the page searching for a special HTML tag indicating a network variable reference. The Web server substitutes the current value of a network variable for this tag when returning information to the browser. Thus, any network variable defined on the *i.LON 100* server can be referenced in a Web page just by incorporating the correct HTML tags.

Web pages may be constructed with any off-the-shelf HTML editor.

Using the *i.LON 100* Server's Web Server

You can create HTML files for the *i.LON 100* Web server may be created with any standard text or HTML editor. The *i.LON 100* Web server supports standard HTML for defining the structure and format of your Web page, as well as the `<ILONWEB>` HTML tag for retrieving dynamic data elements and processing HTML forms. The `<ILONWEB>` tag is an extended HTML tag designed to provide access to the *i.LON 100* server's system and network variable data through a Web browser such as Internet Explorer 6 or better.

HTML files reside in a special directory on the *i.LON 100* server's flash disk. Other related files such as graphics and Java applets may also be stored on the flash disk. Approximately 20MB of space is available for user files on the *i.LON 100* server's flash disk. Files are read and written to the *i.LON 100* server's flash disk using standard FTP over the IP connection.

An *i.LON 100* server hosted Web page can expose any of the data points on the *i.LON 100* server. See Chapter 2 for more information about data points. The following tutorial demonstrates how to read and write data points on the *i.LON 100* server.

Required Hardware

You will need an *i.LON 100* Internet Server with a TP/FT-10 channel for this tutorial.

Required Software

You will need the following software for this tutorial:

- LonMaker Integration Tool Release 3.1, Service Pack 2 (or higher) with the LonMaker Basic Shapes Stencil and the *i.LON 100* Shapes Stencil.
- A standard FTP client application such as Internet Explorer.
- Microsoft Internet Explorer 6.0 or higher

Creating The LonMaker Network

This section describes how to create a simple LonMaker network using LonPoint devices and how to create a Web page through which that network can be monitored and controlled using an *i*.LON 100 server. Please refer to the LonMaker and LonPoint documentation for more information on performing the various LonMaker tasks described in this tutorial.

1. Set the *i*.LON 100 server's IP address, subnet mask, gateway, FTP user name, and FTP password using the *i*.LON 100 Web pages as described in *i.LON 100 User's Guide: Installing, Connecting, and Configuring the i.LON 100*
2. Create and open a new network using the LonMaker tool. This tutorial requires you to be attached to the network.
3. Drag an *i*.LON 100 device shape from the *i*.LON 100 Shapes stencil and name it "*i*.LON 100." Create and commission the *i*.LON 100 device.
4. Drag a **Alarm Generator** shape from the *i*.LON 100 stencil to the drawing. Associate it with the **Alarm Generator[0]** functional block on the *i*.LON 100 server.

Creating Web Pages

To create a simple Web page that will monitor and control the **nviAgCompare** input network variable and the **nvoAnAlmFlg** output network variable on the **Alarm Generator** functional block, follow these steps:

1. Use a text or HTML editor to input the following HTML code:

```
<html>
<head>
<title>Alarm Generator Status</title>
</head>
Setpoint = <iLonWeb FUNC=ShowValue
SYMBOL=NVL_nvoAgAlarmFlag[0]></iLonWeb><p>
</html>
```

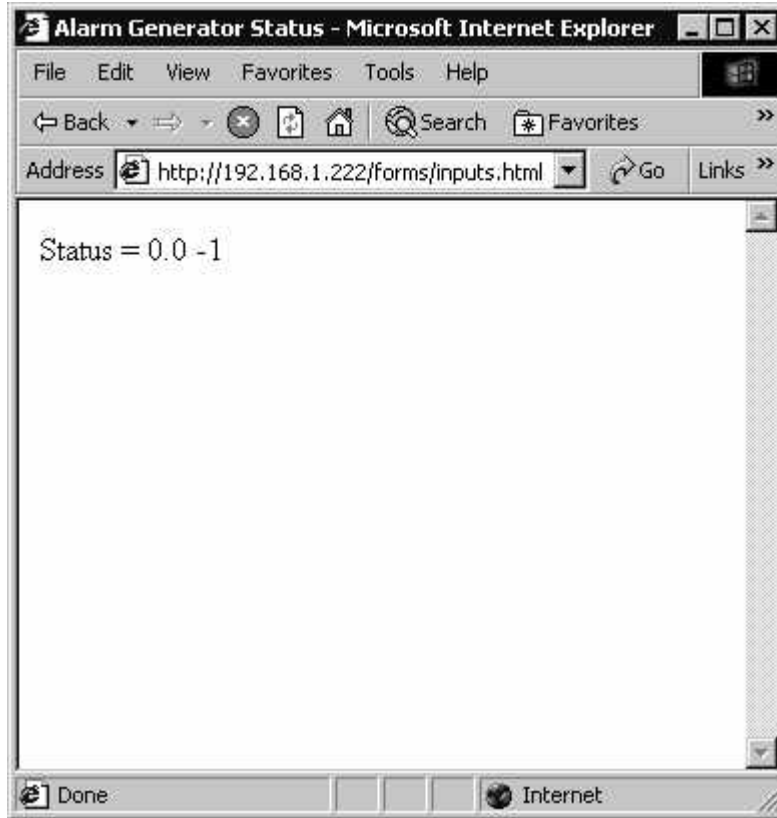
The *i*.LON 100 server's built-in Web server understands the meaning of the special `<iLonWeb>` HTML tag. When the server returns a page to a requesting browser, the **server** parses the page and substitutes the current value of the network variable for the `<iLonWeb>` tag. (Those of you familiar with ASP or other server side substitution technologies will recognize this technique.)

This Web page displays the current value of **nvoAgCompare**.

2. Save the HTML text above as `inputs.htm`.
3. Upload `inputs.htm` to the *i*.LON 100 server's flash disk using a standard FTP program such as Internet Explorer 6 or later.

All Web pages must be in the directory named `Web` or in a subfolder of `Web`. Any page that references network variables must be placed in the `/root/Web/forms` directory or a subdirectory below `/forms`. Do not place non-html files in the `/forms` directory; you may create other directories under `/Web` to store graphics and other content.

- To retrieve the Web page enter `http://24.1.7.251/forms/inputs.htm` (where 24.1.7.151 is the *i*LON 100 server's IP address) in the browser's URL window. You do not need to include the Web directory in the URL. Web is the implied root for all HTTP requests. The Web page displays as shown in the following figure:



Web Page Displaying nvoAgAlarmFlag

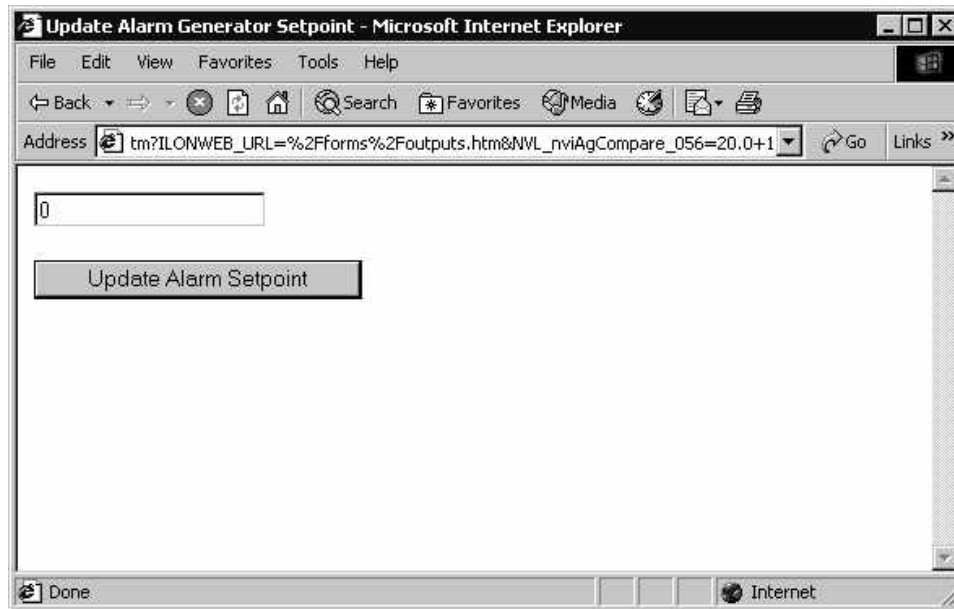
Notice that the *i*LON 100 server's Web server converts the network variable values to strings. The browser "sees" only text.

- Open a new file using a standard text or HTML editor and enter the following HTML code:

```
<html>
<head>
<title>Output Test</title>
</head>
<form action="outputs.htm" method="get">
<iLonWeb_url>
<iLonWeb_func=TextField type=text
symbol=NVL_nviAgCompare_056 size="20"></iLonWeb>
<p>
<input type="submit" value="Update Alarm Setpoint">
</form>
</html>
```

This HTML allows the output network variables defined on the *i*.LON 100 server's Web Server functional block to be controlled from a Web page. Since the NVL_nvoAgCompare data point controls the setpoint for the alarm generator, this will allow you to change the setpoint from the Web page. This code is described in detail in *How the HTML Code Works*, below.

6. Save this code as a file named `outputs.htm`. (The file name is important since the code references the file name in the *action* attribute of each form.) Using an FTP program, transfer the file to the *i*.LON 100 server's `Web/forms` directory, as you did for `inputs.htm`.
7. Enter `http://24.1.1.7.251/forms/outputs.htm` (where 24.1.1.7.251 is the *i*.LON's IP address) in the browser URL window to display the Web page shown in the following figure:



Web Page Displaying Output Network Variable

Enter 10 in the text box and click **Update Alarm Setpoint**. The NVL_nviAgCompare_056 data point is updated; this in turn updates the nviAgCompare network variable on the Alarm Generator functional block.

How the HTML Code Works

Let's examine how this process works. Most Web servers have the ability to call utility programs through a CGI gateway. These utility programs usually do something simple like create a GIF image of a stock chart based on stock symbol provided by the user. Usually, the user types a stock symbol into a text box. When the user clicks the **Create a Chart** button, the stock symbol is passed to the chart-making program as a parameter. The chart-making program makes the chart, saves it as a .GIF file, and tells the server that it has completed its task. The server then serves a page back to the browser, referencing the newly created GIF file.

Updating an output data point uses a similar mechanism. When you click the **Update Alarm Setpoint** button, you are submitting the content of the text box (really the entire form) to the *i*.LON 100 server's Web server. The server passes

the form on to the form-processing engine through an internal *iLON* 100 server interface that is similar to *CGI*. The engine looks for the data point associated with the text box, and then passes the value in the text box along with the data point name to the *iLON* 100 server's data server. The data server then propagates value to the network variable associated with the data point.

```
<form action="outputs.htm" method="get">
```

This tag indicates the beginning of an HTML form. Forms may contain several elements. For example, this form contains a textbox, and a submit button.

All forms have an *action* attribute. This attribute indicates which page will be served next (after submittal). The *method* attribute indicates that the form-processing engine should get all the values in the current form as a *QueryString* when the submit button is pressed. The "post" method is not supported.

```
<iLonWeb_url>
```

The `<iLonWeb_url>` tag allows the *iLON* 100 server to implement the necessary security to prevent access to network variables on Web pages that are outside of a user's access range. See Chapter 8 for more information on setting up *iLON* 100 server Web page security)

```
<iLonWeb func=TextField type=text symbol=NVL_nviAgCompare_056  
size="20"></iLonWeb>
```

Creates an element in a form that is associated with a local data point. In this case the element is a textbox and the data point is `NVL_nviAgCompare_056` (`nviAgCompare_056` is the programmatic name of the `nvo1` network variable – see *iLON 100 User's Guide: Configuring the iLON 100 Applications Using the iLON 100 Configuration Plug-in* for more information)

```
<input type="submit" value="Update Alarm Setpoint">
```

Standard HTML submit button. Every form must have some mechanism that indicates the user has finished filling out the information in the textbox(s) and it is time to send the information to the Web server. The most common method for doing this is to use a submit button. The Web browser recognizes that when a submit button click event occurs, it is time to send the form to the Web server (the *iLON* 100 server in our case) to process the form.

```
</form>
```

Indicates the end of an HTML form.

When the *iLON* 100's Web server serves `outputs.htm` the above HTML code is translated to look like the following:

```
<form action="outputs.htm" method="get">  
<INPUT TYPE=HIDDEN NAME=iLonWeb_URL VALUE=/forms/outputs.htm>  
<INPUT TYPE=text NAME=NVL_nviAgCompare_056 VALUE="0"  
MAXLENGTH=31 SIZE="20" >  
<p>  
<input type="submit" value="Update Alarm Setpoint">  
</form>
```

You can see the effect of the translation by viewing the source code in the browser. Notice that `<iLonWeb_url>` has been translated to `<INPUT TYPE=HIDDEN NAME=iLonWeb_URL VALUE=/forms/outputs.htm>`. The form-processing engine decides which network variables to update based on the hidden field. The form-processing engine will update all the network variables on the page listed in the *value* attribute of this hidden field. Because all data points are sent to the engine when the form is processed, it generally makes sense to have only one form per Web page.

Using The Web Server Functional Block

The *i*LON 100 server contains a Web Server functional block. You can create dynamic network variables on this functional block and connect them to network variables on other devices in order to monitor and control these devices using the *i*LON 100 server. However, the data points that you display in your Web pages are not limited to the dynamic network variables on these blocks. Any data point on the *i*LON 100 may be referenced via an `<ILONWEB>` HTML tag. These blocks are provided to help you organize data points in your LonMaker drawing that are controlled solely via the Web. The following tutorial demonstrates how to create a simple LonMaker network and monitor and control two LonPoint devices using this method.

Required Hardware

You will need the following hardware for this tutorial:

- One *i*LON 100 Internet Server (with A TP/FT-10 channel)
- One LonPoint DI-10 Digital Interface Module
- One LonPoint DO-10 Digital Output Interface Module

Required Software

You will need the following software for this tutorial:

- LonMaker Integration Tool version 3.1, service pack 2 (or higher) with the LonMaker Basic Shapes Stencil, the LonPoint Shapes Stencil, and the *i*LON 100 Shapes Stencil.
- A standard FTP client application such as Internet Explorer.
- Microsoft Internet Explorer 6.0 or higher

Setting Up The Hardware

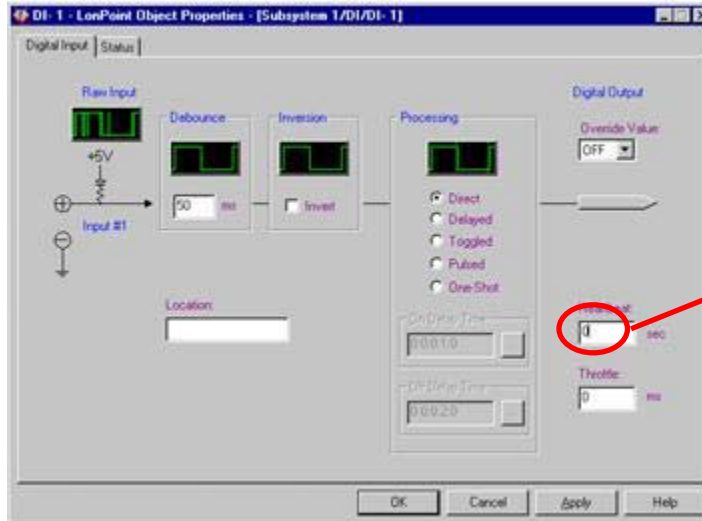
Physically attach the DI-10, DO-10, *i*LON 100 server, and computer running the LonMaker tool to the TP/FT-10 channel. You can use the *i*LON 100 server as an RNI (see Chapter **Error! Reference source not found.**) or use another LONWORKS interface such as a PCLTA-20 or PCC-10. Attach the *i*LON 100 server to the 10BaseT network.

Creating the LonMaker Network

This section describes how to create a simple LonMaker network using LonPoint devices and how to create a Web page through which that network can be monitored and controlled using an *i*LON 100 server. Refer to the LonMaker and

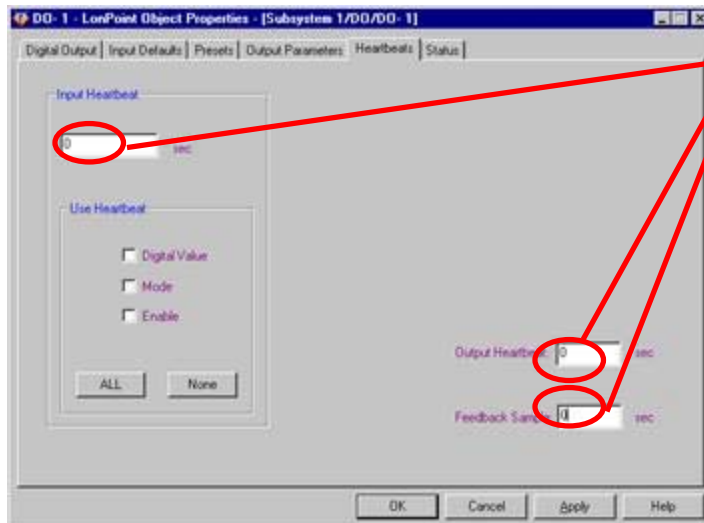
LonPoint documentation for more information on performing the various LonMaker tasks described in this tutorial.

1. Set the *i*LON 100 server's IP address, subnet mask, gateway, FTP user name, and FTP password using the *i*LON 100 Web pages as described in the *i*LON 100 User's Guide: *Installing, Connecting, and Configuring the iLON 100*.
2. Create and open a new network using the LonMaker tool. This tutorial requires you to be attached to the network.
3. Drag a DI-10 device shape and a DO-10 device shape from the LonPoint Shapes stencil onto the LonMaker drawing and commission them. Set their state to OnLine.
4. Drag an *i*LON 100 device shape from the *i*LON 100 Shapes stencil and name it "*i*LON 100". Create and commission the *i*LON 100 device.
5. From the LonPoint stencil, drag four "Digital Input" functional blocks to the LonMaker drawing. Associate one with each input on the DI-10 device.
6. From the LonPoint stencil, drag four "Digital Output" functional blocks to the LonMaker drawing. Associate one with each output on the DO-10 device.
7. Select each LonPoint functional block, right-click it, and then select **Configure** from the shortcut menu. Disable heartbeats in both the input and output functional blocks as shown in the following figures. If you leave heartbeats on, it will be much more difficult to determine if the Web pages that you create later in this tutorial work as intended.



Set this value to zero for all **Digital Input** functional blocks.

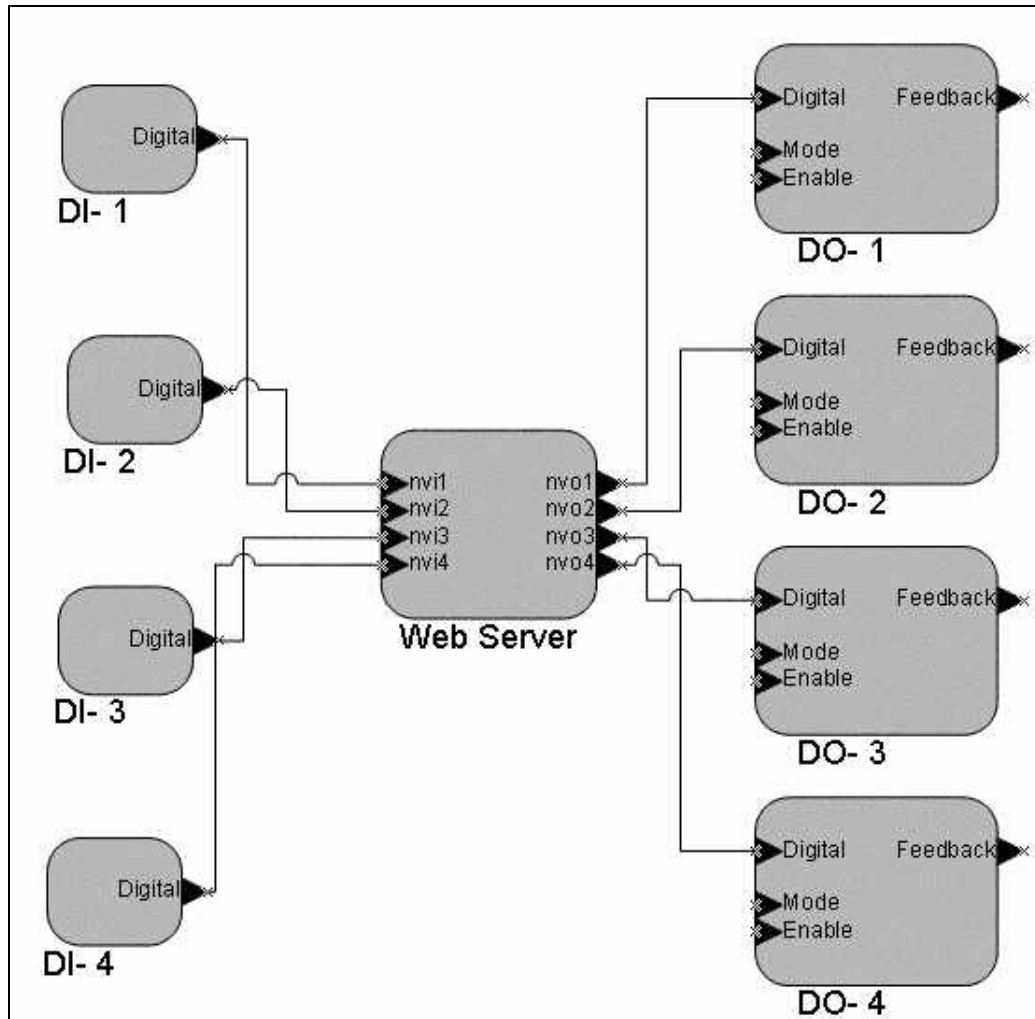
Disable Digital Input Functional Block Heartbeat



Set these values to zero for each **Digital Output** functional block.

Disable Digital Output Functional Block Heartbeat

8. Drag a **Web Server** shape from the *i*.LON 100 stencil to the LonMaker drawing. Associate the functional block with the Web Server[0] functional block and name the functional block **Web Server**.
9. Dynamically create four input network variables on the **Web Server** functional block. Use the **Digital** output network variables on the DI-10 functional blocks as a basis for these network variables. Name these network variables *nvi1*, *nvi2*, *nvi3*, and *nvi4*.
10. Dynamically create four output network variables on the **Web Server** functional block. Use the **Digital** input network variables on the DO-10 functional blocks as a basis for these network variables. Name these network variables *nvo1*, *nvo2*, *nvo3*, and *nvo4*.
11. Connect the **Digital Input** functional blocks to the **Web Server** inputs, and the **Digital Output** functional blocks to the **Web Server** outputs, as shown in the following figure:



The Web Server functional block that you created can be used to monitor and control the network variables on the DI-10 and DO-10 devices through a Web browser. Next, you will write some HTML code and make references to the network variables on an *i*LON 100 server's **Web Server** functional block.

Creating Web Pages

To create a simple Web page that will monitor and control the values of the network variables connected to the Web Server functional block (see *Creating the LonMaker Network*, earlier in this chapter), follow these steps:

1. Use a text or HTML editor to input the following HTML code:

```
<html>
<head>
<title>Display Digital Sensors (inputs)</title>
</head>
nvi1 = <iLonWeb FUNC=ShowValue
SYMBOL=NVL_nvi1_199></iLonWeb><p>
nvi2 = <iLonWeb FUNC=ShowValue SYMBOL=NVL_
nvi2_199></iLonWeb><p>
```



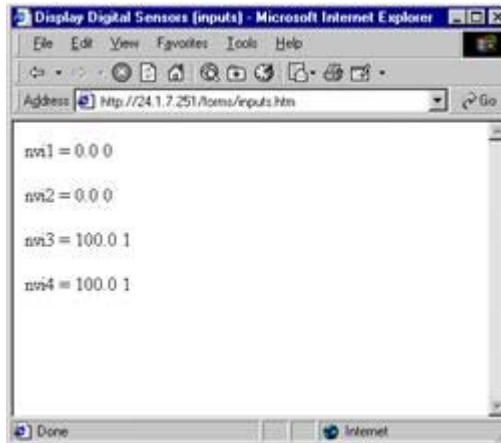
```

nvi3 = <iLonWeb FUNC=ShowValue SYMBOL=NVL_
nvi3_199></iLonWeb><p>
nvi4 = <iLonWeb FUNC=ShowValue SYMBOL=NVL_
nvi1_199></iLonWeb><p>
</html>

```

2. Save the HTML text above as `inputs.htm`.
3. Upload `inputs.htm` to the *iLON 100* server's flash disk using a standard FTP program such as Internet Explorer 6 or later.
4. To retrieve the Web page enter `http://24.1.7.251/forms/inputs.htm` (where 24.1.7.151 is the *iLON 100* server's IP address) in the browser's URL window. You do not need to include the Web directory in the URL. Web is the implied root for all HTTP requests.

Assume that switches 1 and 2 are Off, and switches 3 and 4 are On. The HTML code entered in Step 1 generates the Web page shown in the following figure:



Web Page Displaying Input Network Variable Values

The *iLON 100*'s Web server converts the network variable values to strings. The browser receives only text.

5. Open a new file using a standard text or HTML editor and enter the following HTML code:

```

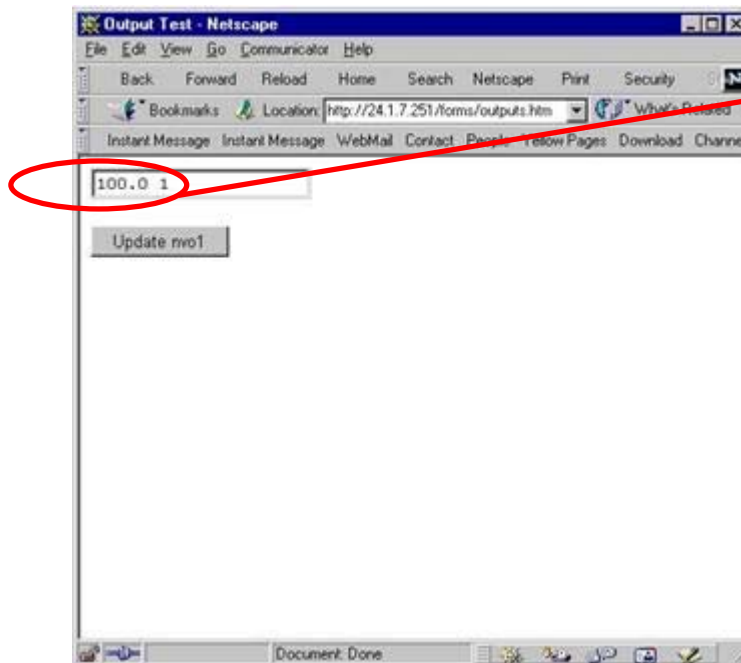
<html>
<head>
<title>Output Test</title>
</head>
<form action="outputs.htm" method="get">
<iLonWeb_url>
<iLonWeb func=TextField type=text symbol=NVL_nv01_199
size="20"></iLonWeb>
<p>
<input type="submit" value="Update nv01">
</form>

```

</html>

This HTML allows the output network variables defined on the *i*LON 100's Web Server functional block to be controlled from a Web page. Since these network variables are connected to the Digital input network variables of the DO-10 device, this will allow you to change the output of the DO device from the Web page.

6. Save this code as a file named `outputs.htm`. (The file name is important since the code references the file name in the *action* attribute of each form.) Using an FTP program, transfer the file to the *i*LON 100 server's `Web/forms` directory, as you did for `inputs.htm`.
7. Enter `http://24.1.1.7.251/forms/outputs.htm` (where `24.1.1.7.251` is the *i*LON's IP address) in the browser URL window to display the Web page shown in the following figure:



Manually enter "100.0 1" (the SNVT_switch value for ON) in the text box and click the **Update nvo1** button. This causes the *i*LON to propagate its network variables.

Web Page Displaying Output Network Variable

Enter `100.0 1` in the text box and click **Update nvo1**. The `NVL_nvo1_199` data point is updated; this in turn updates the `nvo1` network variable on the Web Server functional block.

<iLonWeb> Web Tag Format

The `<iLonWeb>` tag identifies variables, include files, and form elements used by the *i*LON 100 Web server. The general syntax of the `<iLonWeb>` tag is:

```
<iLonWeb FUNC=function SYMBOL={NVL_ | NVE_ | ILON_ | } symbolname >
</iLonWeb>
```

The content of the Web tag is defined by the *function* specified within the <iLonWeb> tag statement. Depending on the function, the Web tag can simply display a network variable value, or it can accept user input from a Web page displayed in a browser and change the value of a network variable. The appropriate network variable or internal i.LON 100 server system variable is specified within the <iLonWeb> tag using the SYMBOL attribute. The remaining attributes further qualify the behavior of the function, such as formatting the output or displaying a specific field within a network variable. These are described in further detail in this chapter.

While the i.LON 100 Web server parses and replaces the Web tags in its HTML files according to their function, it ignores all text between the <iLonWeb.> and </iLonWeb> tags. In contrast, a standard browser reading the same HTML file will ignore the Web tags and process the text between the tags. This feature can be useful when prototyping i.LON 100 Web pages in standard HTML editors. See *Working with Forms* later in this chapter for an example.

FUNC Attribute

Func=ShowValue

The ShowValue function displays the value of the variable identified with the SYMBOL attribute. The syntax of the ShowValue function is the following:

```
<iLonWeb FUNC=ShowValue SYMBOL=symbolname> </iLonWeb>
```

The following example shows the HTML required to display the current value of an input network variable named **localDigitalIn1** using the default text formatting. The network variable name is prefixed with “NVL_” indicating it is a local network variable. The i.LON 100’s Web server uses the first few characters of the symbol name parameter to determine how the symbol is to be acquired. See the *Web Tag Symbol Names* section for more detail on symbol prefixes.

```
<html>
<body>
LocalDigitalIn1=<iLonWeb FUNC=ShowValue
SYMBOL=NVL_localDigitalIn1_199></iLonWeb>
</body>
</html>
```

FUNC=Include

The Include function allows you to include the contents of files. This function reads the contents of the file identified with the FILE attribute. The FILE attribute must specify the include path beginning with “/forms/”. The include file can also contain <iLonWeb> tags. The syntax of the Include function is the following:

```
<iLonWeb FUNC=Include FILE=incpath> </iLonWeb>
```

The following example shows the Include function used in HTML to include the file called **localConfig.htm**.

```

<html>
<body bgcolor="#CCCCCCFF">
<iLonWeb func=Include FILE=/forms/localConfig.htm></iLonWeb>
</body>
</html>

```

FUNC=CreateSymbol

The CreateSymbol function creates a global variable on the Web server. You can use global variables for a number of tasks including passing the variable to another form, modifying its value, or using it to set up a counter. CreateSymbol is often used to pass variables between forms to use with Java scripts. The syntax of the CreateSymbol function is the following:

```

<iLonWeb FUNC=CreateSymbol SYMBOL=User Symbol VALUE=user defined>
</iLonWeb>

```

User Symbol may be any name not starting with symbol prefixes used in the *Web Tag Symbol Names* section.

The following HTML example creates and tests a user-defined symbol called **sValue**. The CreateSymbol function is used to create the symbol, and the ShowValue function is used to display the value assigned to the symbol when it was created.

```

<html>
<body bgcolor="#CCCCCCFF">
<iLonWeb func=CreateSymbol symbol=sValue value="This is a
Test"></iLonWeb>
<p>ShowValue of sValue is <iLonWeb func=ShowValue
symbol=sValue></iLonWeb>
</body>
</html>

```

Form element functions include CheckBox, RadioButton, TextField, and TextArea. See *Working with Forms*, in this chapter for more information.

SYMBOL Attribute

The *iLON 100's* Web server uses Web tag symbol names to identify and retrieve different types of *iLON 100* system data and network variable data that can be displayed in Web pages. Web tag symbol names have prefixes that determine their type as described in the following table.

Web Tag SYMBOL Name Prefixes

Web Tag Symbol Name Prefix	Description
NVE_, NVL_, NVC	Data point type. NVE_ refers to external data points associated with network variables on remote devices that are polled by the <i>iLON 100</i> server. NVL_ refers to local

	data points associated with network variables local to the <i>i</i> LON 100 Web server. NVC_ refers to constant data points created on the <i>i</i> LON 100 server. See Chapter 2 for more information about data points.
ILON_	Refer to system variables generated inside the <i>i</i> LON 100 system software. These values cannot be changed through HTML forms; you can change the values through the <i>i</i> LON 100 console application or the Configuration Web pages, depending on the symbol.
User Defined	Other prefixes that can be used as local Web tag symbols in forms once created with the CreateSymbol tag.

Data Point Symbols (NVL_, NVE_, and NVC_ Prefixes)

You can use three types of data point references in *i*LON 100 Web server Web pages: local data point references, external data point references, and constant data point references.

Local data point references cause the *i*LON 100 to return the value of the network variable associated with the local data point (assuming that something is bound to the local network variable) and allow event-driven updates to be reported on Web pages. Reading and writing of local data points is accomplished through the use of the NVL_ symbol, described below.

You can create external data point references using the *i*LON 100 Configuration Plug-in. The network variable associated with an external data point is not located on the *i*LON 100 server, but exists elsewhere in the LONWORKS network. You have read-only access to external data points associated with output network variables and read-write access to external data points associated with input network variables. Reading and writing of external network variables is accomplished through the use of the NVE_ symbol, described below.

You can create constant data point references using the *i*LON 100 Configuration Plug-in. A constant data point is not associated with a network variable but with a fixed value contained in the *i*LON 100. You have read-write access to constant data points. Reading and writing of constant data points is accomplished through the use of the NVC_ symbol, described below.

Local Data Point Symbols (NVL_ Prefix)

In general, it is best to use local data points as much as possible for reduced network traffic and simplified maintenance. For example, if an output network variable is bound to the *i*LON 100 server, the *i*LON 100 server will get value updates for the remote output whenever its value changes. The Web server can query the value of the local data point without producing any LONWORKS network traffic. If the same network variable is monitored through an external data point (NVE_), a value query message will be sent by the data server on a regular interval (see Chapter 2 for more information).

Local data point symbol names are prefixed with NVL_ when specified in the <iLonWeb> tag. For example:

```
<iLonWeb Func=ShowValue SYMBOL=NVL_nvitemp_199> </iLonWeb>
```

Since the network variable is defined locally, the Web server is aware of its type, size, and format. The syntax for a local network variable reference is the following:

NVL_*nvProgrammaticName*

The *nvProgrammaticName* is the programmatic name of the network variable on the *iLON 100* server.

For example, the action of the following ShowValue function will display the contents of the **DigitalOut** local network variable.

```
<iLonWeb FUNC=ShowValue SYMBOL=NVL_DigitalOut_199> </iLonWeb>
```

External Data Point Symbols (NVE_ Prefix)

While it is normally advantageous to use local data points, there are several cases where this method does not work:

- Your network tool does not know how to create dynamic network variables.
- Your system is pre-installed, or self-installed and thus has no network tool.
- Your system is installed on the zero-length domain.
- You can create dynamic network variables, but you need so many that you run out of address table entries.

In these cases, you need to poll the network variables on the network without binding them to network variables on the *iLON 100* server. Because the network variable is not local to the *iLON 100* server, the external data point must be explicitly created. You can create external data points for any network variables in the network by creating an XML NVE driver file; this can be done using the *iLON 100* Configuration Plug-in as described in Chapter 2 or manually as described in the *iLON 100 Programmer's Reference*. This approach has the following disadvantages:

- You must manually assign addresses for the external data points.
- You must manually update the addresses if they change for any reason, such as a change in your network configuration.

External data point symbol names are prefixed with NVE_ when specified in the **<iLonWeb>** tag. For example:

```
<iLonWeb Func=ShowValue SYMBOL=NVE_nviSwitch> </iLonWeb>
```

The *iLON 100* data server will look up the name of the external data point in the NVE driver file and serve the appropriate value to the Web server.

When an external data point is created, its name appears in the **External Points** tab of the *iLON 100* Configuration Plug-in. This name must be used when specifying the external data point in the **<iLonWeb>** tag. External data point names are always prefixed with NVE_.

Constant Data Point Symbols (NVC_ Prefix)

Constant data point symbol names are prefixed with NVC_ when specified in the **<iLonWeb>** tag. For example:

```
<iLonWeb Func=ShowValue SYMBOL=NVC_myConstant> </iLonWeb>
```

The *iLON 100* data server will look up the name of the constant data point in the NVC driver file and serve the appropriate value to the Web server.

When a constant data point is created, its name appears in the **Constants** tab of the *iLON 100* Configuration Plug-in. This name must be used when specifying

the constant data point in the <iLonWeb> tag. Constant data point names are always prefixed with NVC_.

System Symbols (ILON_ Prefix)

System symbol names are specified in the <iLonWeb> tag with the SYMBOL= attribute to display i.LON 100 server information. System symbol names are prefixed with “ILON_.”

The system symbols in the following table are used with the ShowValue tag to display information from the i.LON 100 server.

System Symbol Names (ILON_ Sys) Used With ShowValue

Web Tag Symbol	Description
iLon_Sys_IpAddress	The IP address of the i.LON 100 server, <i>e.g.</i> 10.1.253.101.
iLon_Sys_IpMask	The IP subnet mask of the i.LON 100 server, <i>e.g.</i> 255.255.255.0.
iLon_Sys_IpName	The hostname name of the i.LON 100 server, <i>e.g.</i> iLonDataSvr01
iLon_Sys_Gateway	The IP address of the gateway device, <i>e.g.</i> 10.1.253.1
iLon_Sys_DhcpEnabled	1 if DHCP is enabled, 0 [zero] if not enabled.
iLon_Sys_CurrentIpAddress	The current IP address. If you change the IP address, the change will not take place until the i.LON 100 server is reset.
iLon_Sys_CurrentIpMask	The current IP subnet mask. If you change the subnet mask, the change will not take place until the i.LON 100 server is reset.
iLon_Sys_CurrentGateway	The IP address of the current gateway device. If you change the gateway, the change will not take place until the i.LON 100 server is reset.
iLon_Sys_MacAddress	The Ethernet MAC address of the i.LON 100 server, <i>e.g.</i> 00-23-34-45-56-AB
iLon_Sys_DNSServer1	The IP address of the first DNS Server.
iLon_Sys_DNSServer2	The IP address of the second DNS Server.
iLon_Sys_IpDomain	The domain name for the DNS server.
iLon_Sys_LtUids	The Neuron IDs available in the i.LON 100 server for LONWORKS applications.

Web Tag Symbol	Description
iLon_Sys_LtXcvrId	The LONWORKS transceiver ID of the attached transceiver.
iLon_Sys_HttpPort	The IP port used by the LONWORKS IP communications software. Normally 1628.
iLon_Sys_ConfigServer	The IP address and port of the Configuration Server, if any. The port follows the address after a colon, <i>e.g.</i> 10.1.253.34:1629
iLon_Sys_TimeServer1	The IP address and port of the first time server, <i>e.g.</i> 10.1.0.1:123
iLon_Sys_TimeServer2	The IP address and port of the second time server, <i>e.g.</i> 10.1.0.1:123
iLon_Sys_TimeSynched	1 for synchronized with a server, 0 for not synchronized.
iLon_Sys_TimeLastSynched	The time in Unix format at which the time was last synchronized with a time server.
iLon_Sys_Date	The date in Unix format of the local date on the <i>i</i> LON 100 server.
iLon_Sys_Time	The time in Unix format of the local time on the <i>i</i> LON 100 server.
iLon_Sys_TimeZone	The time zone settings for the <i>i</i> LON 100 server. This includes both the offset from universal coordinated time (UTC) and the settings for daylight savings time.
iLon_Sys_LONWORKS_Addr	The domain/subnet/node address of the <i>i</i> LON 100 server in the LONWORKS network.
iLon_Sys_FtpUsername	The FTP username for the <i>i</i> LON 100 server.
iLon_Sys_FtpPassword	The FTP password for the <i>i</i> LON 100 server.
iLon_Sys_AutoAnswer	1 if dial-in to the <i>i</i> LON 100 server is enabled. 0 if it is not.
iLon_Sys_TelnetEnable	1 if telnet access to the <i>i</i> LON 100 server console application is enabled; 0 if it is not.
iLon_Sys_FtpEnable	1 if FTP access to the <i>i</i> LON 100 server is enabled; 0 if it is not.
iLon_Sys_RniEnable	1 if RNI access to the <i>i</i> LON 100 server is enabled; 0 if it is not.

Web Tag Symbol	Description
iLon_Sys_SOAPEnable	1 if SOAP access to the <i>i</i> .LON 100 server is enabled; 0 if it is not.
iLon_Sys_AuthKeyRaw	The raw 16 byte MD5 authentication key.
iLon_Sys_AuthKeyHashed	The hashed 16 byte MD5 authentication key.
iLon_Sys_LtUids	The Neuron ID of the <i>i</i> .LON 100 server.
iLon_Sys_CENELECEnable	1 if the CENELEC protocol is enabled on the <i>i</i> .LON; 0 if it is not.
iLon_Sys_LnsServer1	The IP address/host name and port of the primary LNS Server (<i>i.e.</i> 128.45.5.2:2020 or myilon.abccorp.com:2020).
iLon_Sys_LnsServer2	The IP address/host name and port of the secondary LNS Server.
iLon_Sys_LnsServer3	The IP address/host name and port of the tertiary LNS Server.
iLon_Sys_RNIPort	The port on which the <i>i</i> .LON 100 server listens for RNI connections.
iLon_Sys_RniMaxIdleTime	The maximum idle time before the <i>i</i> .LON 100 server terminates an idle RNI connection.
iLon_Sys_MailServer	The IP address/host name and port of the SMTP mail server used to send Alarm Notifier emails.
iLon_Sys_MailLogin	The login in name required by the SMTP mail server.
iLon_Sys_MailPassword	The password required by the SMTP mail server.
iLon_Sys_MailOriginator	The email address that will be seen in the From field of emails sent by the <i>i</i> .LON 100 server.
iLon_Sys_FreeDiskSpace	The amount of free disk space on the <i>i</i> .LON 100 server.
iLon_Sys_DataLogFull	The percentage of the data logging space on the <i>i</i> .LON 100 server that is currently full.
iLon_Sys_TraceFile	The name of the console application trace file.
iLon_Sys_NumNVs	The number of network variables on the <i>i</i> .LON 100 server.

Web Tag Symbol	Description
iLon_Sys_PhoneNumbers	The phone number of the <i>i</i> .LON 100 server.
iLon_Sys_UplinkNVUpdate	1 to initiate an uplink connection on any network variable update; 0 to not do so.
iLon_Sys_UplinkExpMsg	1 to initiate an uplink connection on any explicit message update; 0 to not do so.
iLon_Sys_UplinkExplicitCode	1 to initiate an uplink connection on any explicit message update with an explicit message code in the range defined by iLon_Sys_UplinkExplicitCodeVal; 0 to not do so.
iLon_Sys_UplinkExplicitCodeVal	The range of explicit message codes to be used if iLon_Sys_UplinkExplicitCode is set. The format for this range is <i>xx-yy</i> where <i>xx</i> and <i>yy</i> are integers between 00 and 79 and <i>xx</i> is less than <i>yy</i> .
iLon_Sys_DnsServerViaDhcpEnabled	1 to get the primary DNS server via DHCP; 0 to not do so. Does not get the secondary DNS server.
iLon_Sys_DnsDomainViaDhcpEnabled	1 to get the DNS domain suffix via DHCP; 0 to not do so.
iLon_Sys_RemoteRebootEnable	1 to allow the <i>i</i> .LON 100 to be rebooted remotely; 0 to forbid remote reboots. This value can only be changed in security access mode.
iLon_Sys_SecureAccessAlways	1 to allow properties that normally can only to be changed in security access mode to be changed at any time; 0 to have security access mode function normally. This value can only be changed when in security access mode. That setting this value to 1 makes the <i>i</i> .LON 100 less secure.

Two types of memory for processing Web requests are pre-allocated when the Web server starts: **global memory**, which is used by all the Web tasks, and **per-request** memory, which is used by a particular Web task for the duration for one request. The symbols in the following table allow a page to obtain memory information. For example, in the CreateSymbol function, some of the parameters and security information use the global partition while the request partition is used by the current request only. The global symbols can be placed in any pages after the pages that perform CreateSymbol are used. The request symbols should be placed at the end of pages that process the most complex requests. All of the following symbol values are expressed in bytes.

System Symbol Names (ILON_ Prefix) Used to Obtain Memory Information

Web Tag Symbol	Description
iLon_Mem_RequestPartTotal	Total memory allocated to process requests.
iLon_Mem_RequestPartFree	Free memory in a request partition.
iLon_Mem_RequestPartLargest	Largest free block in a request partition.
iLon_Mem_RequestPartUsed	Used memory in a request partition.
iLon_Mem_GlobalPartTotal	Total global memory allocated.
iLon_Mem_GlobalPartFree	Free memory in a global partition.
iLon_Mem_GlobalPartLargest	Largest free block in the global partition.
iLon_Mem_GlobalPartUsed	Used memory in a global request partition.

Web Tag Attributes

When a network variable is specified with the NVL_ or NVE_ symbol in the SYMBOL= attribute of the <iLonWeb> tag, four optional attributes may be used to further define how a network variable is displayed or changed.

FIELD:

FORMAT:

PROPAGATE:

WAIT:

The attributes may be used in any combination within a single <iLonWeb> tag with the exception of **WAIT:**. **WAIT:** must be used in conjunction with **PROPAGATE:**. The colon (:) is required as part of the syntax. If Web tag attributes are used, use the exclamation point (!) to delimit the attributes. For example:

```
<iLonWeb FUNC=TextField  
SYMBOL=NVL_DigitalOut!FIELD:value!PROPAGATE:TRUE> </iLonWeb>
```

The Web tag in the example above will display the contents of the field called **value** contained in the local network variable, **Digital_Out**. The value will be displayed as a read-write value in a text field. If the form containing this text field is submitted, the new value will be propagated onto the network immediately.

FIELD:

Network variables may contain fields that vary according to the network variable type. The contents of these fields can be displayed and changed by specifying the field name with the **FIELD:** attribute.

For example, the SNVT_switch type contains two fields: **value** and **state**. To display the contents of the **value** field in the local network variable **DigitalOut**, use the following tag.

```
<iLonWeb FUNC>ShowValue SYMBOL=NVL_DigitalOut!FIELD:value>  
</iLonWeb>
```

Important! The default value of the **PROPAGATE**: attribute is False when operating on a network variable field. For further information, see *PROPAGATE*.

FORMAT:

You may assign a format to a network variable to control how the data is displayed and changed in your Web page. Specify the format of the network variable using the **FORMAT** attribute.

Network variable formats can come from the following three places:

Standard Resource File Set

This is a set of files that describes the data structures within standard network variable types (SNVTs) and standard configuration property types (SCPTs) and also describes the formats to be used for display of SNVT and SCPT data. On the *iLON 100*, these files may be found in the `/root/lonworks/types` directory, and are named `STANDARD.ENU`, `STANDARD.TYP`, `STANDARD.FMT`, and `STANDARD.FPT`.

The default format for a SNVT is its native format, as described in the `STANDARD.FMT` text file within the resource file set. To assign a different SNVT or SCPT format, or to assign a SNVT or SCPT format to a user network variable type, explicitly assign the format type. For example:

```
FORMAT:SNVT_switch
```

User Resource File Sets

These are sets of files created by device manufacturers to describe manufacturer-defined user network variable types (UNVTs) and user configuration property types (UCPTs). Using the same mechanisms as the standard resource file set, they describe how to format data from a particular manufacturer's device. On the *iLON 100*, all resource files will be found in the `/root/lonworks/types` directory.

UNVT and UCPT formats can be specified using the format name of the form:

```
[#progamID[selector] .] formatName
```

The program ID is optional; if it is not supplied, the Program ID of the *iLON 100* server will be used. In this syntax, the **bold** “#”, “[”, “]” and “.” characters are literal characters. The program ID is represented as a hex byte string (in the “`RAW_HEX_PACKED`” format described below). The selector is a one-digit string from 0 to 6, and the format name syntax is similar to that used for SNVT and SCPT types, except that the type name starts with “UNVT” or “UCPT” instead of “SNVT” or “SCPT”. For example:

```
FORMAT:#8011223344556677[1].UNVT_switch
```

Built-in Formats

The underlying formatting engine provides built-in formats, including **RAW**, **RAW_HEX**, **RAW_HEX_PACKED**, **FLOAT_TEXT**, **STRING**, and **UNIT_TEXT**. All of these formats display the network variable data byte-by-byte, in the same order that the bytes arrive on the network (the Neuron Chip has big endian byte ordering, the opposite of the Pentium's little endian ordering, and network variable data must be in big endian order on the LONWORKS network). The **RAW** format displays the data as decimal byte values, with each byte separated by a space. The **RAW_HEX** format displays the data as hexadecimal byte values, with each byte separated by a space. The **RAW_HEX_PACKED** format displays each byte as a two-digit hex value, with no spaces in between the byte values.

The **RAW_HEX_PACKED** format is the default format for UNVT network variables that do not specify a format.

PROPAGATE:

The **PROPAGATE:** attribute allows you to specify whether a value should be transmitted across the network immediately, or buffered. For example:

```
<iLonWeb FUNC=TextField  
SYMBOL=NVL_DigitalOut_199!FIELD:value!PROPAGATE:TRUE></iLonWeb  
>
```

The **PROPAGATE:** attribute defaults to **TRUE** when a tag references a complete network variable, and **FALSE** when a tag references a network variable field. Network variable fields may not be manipulated individually over the LONWORKS network—the entire network variable value can be changed, but not one field at a time. These default settings are designed to assure that updates are consistently delivered for the specified network variable. In addition, it is typically more efficient to update several fields and propagate the network variable once, instead of causing propagation for each field update.

If your HTML form sets a network variable value field-by-field, set the **PROPAGATE:** attribute to **False** for all of the network variable field tags in the form except the last one. The last **PROPAGATE:** attribute of **TRUE** will cause the network variable value to be propagated over the network. If you set **PROPAGATE:** to **True** for each field, some fields may contain indeterminate values in the network variable update, which should not be propagated.

WAIT:

The **WAIT:** attribute allows you to specify whether to wait for the acknowledgement of a write command. Using the wait feature, you may test whether a local output network variable's update is acknowledged by the remote inputs. The test will only be valid if the local output network variable is connected to one or more remote network variables by the acknowledged address services. If the connection is made by an unacknowledged message service, the test will always return a positive response, even if the network variable update did not reach its target.

To perform the test, follow these steps:

1. Set the **WAIT:** attribute to TRUE by adding !WAIT=TRUE to the Web tag responsible for writing the network variable. This causes the data server to wait for an acknowledgement (ACK) or failure from the network variable update before returning from processing that field.
2. Modify the Web page so it reads back the results of the acknowledgement. Following the update tag, place a **ShowValue** tag for the same network variable and specify a field name of "\$ErrStatus" (!FIELD:\$ErrStatus). This field returns the value of the network variable's error status.

If all of the expected acknowledgments were received, the returned error string will be empty. If any acknowledgements failed, the string "No acknowledgement from remote network variable" will be returned. Since an empty error string will be returned on success, you can always display the error string, perhaps in a color indicating that an error condition exists.

Working with Forms

Forms are used in Web pages to get information from end users. HTML form elements allow you to present and collect that information by using input objects such as text boxes and check boxes. **Only one form is recommended for each page.** See *Form Element Functions* later in this manual for further information. A function within the *i.LON 100* Web tag, called a *form element function*, invokes a routine that performs a specific task. The form element function acts upon the network variable that you specify in the **SYMBOL=** attribute. Other attributes may further qualify what the function does, such as formatting the output or displaying a specific field within a network variable.

The following Web tag uses the **Check Box** function and assigns a value of 1 to the state field of the DigitalOut network variable when the user selects this checkbox in a Web page.

```
<iLonWeb FUNC=CheckBox SYMBOL=
NVL_DigitalOut_199!FIELD:state></iLonWeb>
```

Opening a Form

To include a form in a Web page, start with a HTML document then insert the appropriate <iLonWeb> tags to build your form. The first step in building a form is to use a form element function to open a form.

```
<FORM ACTION=filename.htm METHOD=GET><iLonWeb_URL></FORM>
```

Web Tag Element	Description
<FORM >	This tag is standard HTML and signals the start of a form.
ACTION=	This attribute is standard HTML and identifies what happens to the data when the form is submitted for processing.

METHOD=	This attribute defines the method used to send data to the server. GET is the supported method and sends data to the server by appending the data to the URL itself after a question mark (?) separator. POST is <i>not</i> supported.
<iLonWeb_URL>	This tag is required between the <FORM> and </FORM> tags. This tag implements a security feature by creating a hidden field that is the URL of the page containing the form. The form processor checks the URL to ensure the users are accessing only the variables that they are authorized to access.
</FORM>	This tag signifies the end of the form.

Important! While it is recommended that you use only one form per page, the *iLON 100* Web server will support multiple forms on the same page on the condition that each read-write network variable defined on the page is defined on one form only. This prevents unintentional updating of network variables.

Submit or Reset a Form

Once information has been entered into a form, the form must be submitted to the server. This is accomplished using the Submit function. Alternatively, a form that has not already been submitted can be restored to its original values using the Reset function. These two functions, Submit and Reset, are standard HTML form elements that control forms in Web pages. For information on their use and syntax, consult an HTML reference.

When used in conjunction with the *iLON 100* Web tags, the Submit function creates a button that, when clicked, informs the *iLON 100* Web server to update those read/write data points whose values are displayed on forms in the page. The Reset function creates a button that resets any form element to its original value, or, if a submit button was clicked, to its most recently submitted value. For example:

```
<INPUT type="submit" value="Write NV">
<INPUT type="reset" value="Reset modifications">
```

Important! Clicking the submit button causes all forms on a page to submit their data to the server, even if no change has been made to a form. While the *iLON 100* Web server will support multiple forms per page, provided each form references a unique data point, it is recommended that only one form be used per page, to avoid the unintentional updating of unchanged data points.

Refresh a Form

After submitting a value to update a data point, the URL is appended with the submitted value in the browser's address window. If you subsequently use the Web browser's Refresh button to try to obtain the current value of the data point, the value appended to the URL is written to the data point instead. Instead of using the Web browser's Reload or Refresh button, create your own Refresh button to obtain the current value of a data point that is defined in your Web page. This function will get the latest value of the network variable and display it when the Web page is reloaded. For example:

```
<INPUT type="button" value="Refresh"
onClick="window.location.assign(window.location.pathname) ">
```

When you click a Refresh button in a form to reload a Web page, the browser might load the Web page from cache memory. That cached Web page could contain old data point values. To force the browser to load a new page, add the following Meta Web tag in the <head> tag of the Web page.

```
<head>
<META HTTP-EQUIV="Expires" CONTENT="Tue, 25 Apr 1995 09:30:00
-0700">
</head>
```

As an alternative to using the Refresh button, create a link to the current Web page:

```
<a href="iLonTest1.htm"><b>Refresh</b></a>
```

Form Element Functions

This section describes the form element functions supported by the *i*LON 100 Web server. The following table lists the *i*LON 100 Web server form element functions and their actions. Unlike most HTML elements, the *i*LON 100 Web tag functions are case sensitive.

Function	Action on the Specified Network Variable
<i>CheckBox</i>	A value of 1 is selected for the specified data point when the user sets the check box, a value of 0 is written if the checkbox is clear.
<i>Hidden</i>	Hides the text field that contains the value of the specified data point.
<i>RadioButton</i>	The value assigned to a radio button is selected for the specified data point when the user clicks it.
<i>TextField</i>	Displays the value of the specified data point with access to modify it.

CheckBox

A *check box* is an object that presents the user with a choice to set or clear an option. The `CheckBox` tag creates a checkbox in your Web page form and sends a value of 1 to the form processor when the user sets the checkbox and submits the form for processing. The form processor will set a value of 0 for a cleared checkbox.

The Web tag example below creates a check box object in a Web page form. The table that follows describes each Web tag element.

```
<iLonWeb FUNC=CheckBox SYMBOL=NVL_nvoCbx1_199></iLonWeb>
```

Web Tag Element	Description
<iLonWeb>	The <i>i</i> LON 100 tag.

FUNC=CheckBox	Specifies the CheckBox function and causes a checkbox to be created in the form.
SYMBOL=NVL_nvoCb1_199	Specifies that the local data point called NVL_nvoCb1_199 is the symbol to which a value is written when the user selects the checkbox in the form.
</iLonWeb>	The ending <i>i</i> LON 100 tag.

Hidden

The Hidden function allows you to include text in your Web page without displaying it on the browser window. This function is useful when you want to make network variable values available to a program, like JavaScript, but not for users to see or edit. For example, the following Web tag hides the value of the NVL_nvoDigital data point on the Web page.

```
<iLonWeb FUNC=Hidden SYMBOL=NVL_nvoDigital></iLonWeb>
```

Web Tag Element	Description
<iLonWeb>	The <i>i</i> LON 100 tag.
FUNC=Hidden	Hides the text field that contains the value of the specified data point.
SYMBOL= NVL_nvoDigital	Specifies the local data point named NVL_nvoDigital on the Web page.
</iLonWeb>	The ending <i>i</i> LON 100 tag.

RadioButton

Radio buttons are objects that present the user with a series of items and allow the user to choose one item from the series. A value is assigned to each radio button defined in the tag, and when the user selects a radio button, the corresponding value is sent to the form processor.

The following Web tag example creates a series of radio button objects in a Web page form. The table that follows describes each Web tag element.

```
<iLonWeb FUNC=RadioButton SYMBOL=NVL_nvoRb1_199 VALUE="1"
Checked></iLonWeb>
<iLonWeb FUNC=RadioButton SYMBOL=NVL_nvoRb1_199 VALUE="2">
</iLonWeb>
<iLonWeb FUNC=RadioButton SYMBOL=NVL_nvoRb1_199
VALUE="3"></iLonWeb>
```

Web Tag Element	Description
<iLonWeb>	The <i>i</i> LON 100 tag.

FUNC=RadioButton	Specifies the radio button function and causes a radio button object to be displayed in the form.
SYMBOL=NVL_nvoRb1	Specifies that the local data point called NVL_nvoRb1 be written to when the form is processed.
VALUE="1"	Specifies the data point update value assigned to the radio button.
Checked	Indicates that this radio button is selected by default.
</iLonWeb>	The ending <i>i</i> .LON 100 tag.

TextField

A text box allows the user to insert a relatively small amount of text information into a form. Use this tag when you want to modify a network variable value.

The text box example and description below creates a text box that is 20 characters wide and allows the user to update a local data point named **NVL-nvoDigital1**.

```
<iLonWeb FUNC=TextField TYPE=text SYMBOL=NVL_nvoDigital1
SIZE=20 MAXLENGTH=58></iLonWeb>
```

Web Tag Element	Description
<iLonWeb>	The <i>i</i> .LON 100 tag.
FUNC=TextField	Specifies the text field function. Causes a text box to be displayed in the Web form.
TYPE={text password}	The Text type specifies the input will appear as text. The password type specifies that input will appear as asterisks in the text box.
SYMBOL=NVL_nvoDigital1	Specifies that the local data point called NVL_nvoDigital1 be written to when the form is processed.
SIZE=20	Indicates that the width of the text box is 20 characters.
MAXLENGTH=58	Indicates that the field is 58 characters long. If SIZE is less than MAXLENGTH, the input text will scroll within the text box displayed in the browser.
</iLonWeb>	The ending tag

Appendix B

Troubleshooting

This appendix can be used to diagnose common problems that occur with the *i.LON 100*.

I have created NVE style points and although I can see them in the plug-in and reference them using the SOAP/XML interface I can not reference them using Web tags.

- *i*LON 100 Web tags do not allow NVE tag names with spaces. Be sure your tag names do not have spaces. Note that the SOAP/XML server and the plug-in do allow spaces under some conditions, but the best practice is never to include spaces in NVE names.

I can access my *i*LON 100 Web pages but some content seems to be missing.

- The *i*LON 100 has been designed to work with Microsoft Internet Explorer 6.0 or later. Some pages will not display correctly on other browsers or prior versions of IE.

I defined a dynamic network variable in the LonMaker tool, but I can't reference it in my Web page. What is wrong?

- The *i*LON 100 will append a number to any network variable you defined in LonMaker. This allows network variable names to remain unique even if two functional blocks have an input named nviMyInput. You can see what number is appended by looking at the data server tab on the *i*LON 100 plug-in.

The *i*LON 100 exhibits problems due to a low-memory condition. This could be indicated by one or more of the following: “out-of-memory” messages, slow network access, application performance problems, or even an unexpected reboot.

- Reduce the number of Alarm Notifiers and/or the size limit of the alarm Summary Logs (which are kept in RAM).
- Reduce the number of Web clients that are simultaneously accessing the *i*LON 100.
- If you are making calls to the SOAP interface on the *i*LON 100, refer to the *i*LON 100 Programmer's Guide for specific recommendations on limiting SOAP messages.