*i.*LON™ 100 *e2* Internet Server Programmer's Reference

Version 1.1

ECHELON®
Corporation

078-0250-01B

Echelon, LON, LonWorks, NodeBuilder, LonTalk, Neuron, LonMark, LNS, LonBuilder, LonUsers, BeAtHome, LonManager, 3120, 3150, LonPoint, Digital Home, LonWorld, ShortStack, *i.*LON, the Echelon logo, and the LonMark logo are registered trademarks of Echelon Corporation. LNS Powered by Echelon, LonMaker, LonLink, LonResponse, OpenLDV, LONews, Open Systems Alliance, Panoramix, Panoramix Powered by Echelon, LonMark Powered by Echelon, Powered by Echelon, and LonSupport 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

Table Of Contents

1 Introduction to the *i.*LON 100 SOAP/XML Interface

The *i.*LON 100 contains a powerful microprocessor with a real-time, multi-tasking operating system that manages its various applications. These applications include alarming, scheduling, data logging and network variable type translation. Generally, you will configure these applications using the *i.*LON 100 Configuration Software, as described in the *i.*LON *100 Internet ServerUser's Guide*. The *i.*LON *100 Internet Server User's Guide* provides instructions to follow when configuring the *i.*LON 100 Configuration Software, as well as general information on the diffferent *i.*LON 100 applications, and guidelines to follow when using these applications.

You can also use the SOAP (Simple Object Access Protocol) / XML (Extensible Markup Language) interface provided with the *i.*LON 100 to configure and use these applications. XML is a universal format used to deliver data through structured documents over the Web. It allows developers to store data for any application in a standard, consistent way. SOAP is an interface that provides a mechanism for different applications and devices to communicate with each other over the Internet, regardless of platform, by sending SOAP messages to each other. SOAP relies on XML to define the format and contents of its messages.

The configuration of each *i.*LON 100 application is stored in an XML file. The *i.*LON 100 reads these files during its boot process, and sets the operating parameters of each application based on the configuration data contained in the XML file for that application.

The *i.*LON 100 includes a set of SOAP functions that you can use to create and manage the configuration of each application. Each time you invoke any of these functions, a SOAP message is sent to the *i.*LON 100. The content of the SOAP message is based on the input you supply to the function. The *i.*LON 100 reads the contents of the message, writes the contents of the message to the applicable XML file, and adjusts the operating parameters of its applications accordingly. All of this occurs while the *i.*LON 100 is operating.

It is important to note that the XML files described in this document store the configurations of the *i.*LON 100 applications. They do not store the data generated by these applications. The data generated by the *i.*LON 100 applications is stored in the flash memory of the *i.*LON 100.

However, this does not mean that application configuration is the only task you can perform with the *i.*LON 100 SOAP/XML interface. The SOAP/XML interface also includes functions you can use to access, read and analyze the data generated by the *i.*LON 100 applications. And so you can use the SOAP/XML interface not only to configure the various applications of the *i.*LON 100, but to monitor them as well.

1.1 About This Document

This document describes the XML files that store the configurations of the various *i.*LON 100 applications, and the SOAP functions you can use with each application. The SOAP interface provided with the *i.*LON 100 conforms to the SOAP 1.1 proposed Technical Recommendation:

http://www.w3.org/TR/2000/NOTE-SOAP-20000508

This document also describes how to configure the *i.*LON 100 applications by manually creating and modifying the XML configuration files. Once you have created the XML files, you can download them to the *i.*LON 100 via FTP. The *i.*LON 100 will read the downloaded files and adjust its operating parameters accordingly the next time it is rebooted.

You can create or modify the files using any XML editor or ASCII text editor. This document provides examples you can use when creating the XML configuration files for your *i.*LON 100, and instructions to follow when downloading these files to the *i.*LON 100. The XML files used by the *i.*LON 100 applications conform to the XML 1.0 Technical Recommendation:

http://www.w3.org/TR/2000/REC-xml-20001006

Echelon strongly recommends that you use the SOAP interface to configure the applications of your *i.*LON 100. The *i.*LON 100 performs error-checking on all data written in a SOAP message, so that invalid data will not be written to any of the XML files. The *i.*LON 100 will not perform error-checking on any XML files downloaded to it via FTP, and so manually editing the XML files may cause errors during the boot process. Additionally, you can send SOAP messages to the *i.*LON 100 while it is operating, and the *i.*LON 100 will update the XML files affected by the SOAP messages without requiring a reboot.

You may find the information in this document that pertains to manually creating and managing XML files useful if you are using several *i.*LON 100s, and would like to use the same configuration on each one. In that case, you could configure one of the *i.*LON 100s, copy its XML files, and download them to the appropriate directories of the other *i.*LON 100s to obtain the same configuration in all of them. For more information on how to download XML configuration files, see *Copying XML Files Between i.LON 100s* on page 15-2.

1.2 Programming Samples

This document includes programming samples written in Microsoft Visual Basic .NET ® to illustrate concepts described in this manual. To make these samples more easily understood, they have been simplified. Error checking has been removed, and in some cases, the examples are only fragments that may not compile without errors or warnings.

1.3 Getting Started

You should review Chapter 2*, SOAP Messages and the i.LON 100 WSDL File ,* before proceeding to the rest of this document and learning about the functions and applications of the SOAP/XML interface. Chapter 2 describes the WSDL file which defines the *i.*LON 100 SOAP/XML interface, and contains vital information you will need to know before referencing the WSDL file and using the various functions of the SOAP/XML interface. The final section of Chapter 2, *Writing SOAP Applications*, sets a roadmap you can follow when reading through the rest of the document.

If you are upgrading to version 1.1 of the SOAP/XML interface, it is also recommended that you review the next section, *i.LON 100 Version 1.1 SOAP/XML Interface Upgrades,* before proceeding. This section outlines the changes that have been made to the SOAP/XML interface for version 1.1.

1.4 *i.*LON *100 Version 1.1 SOAP/XML Interface Upgrades*

This section provides an overview of the changes made to the SOAP/XML interface for version 1.1, and includes the following sections:

- Modified SOAP Applications and Functions
- Changes to SOAP Message Formats

You may find these changes advantageous when using version 1.1 of the SOAP/XML interface. You should also note that version 1.1 provides complete compatibility with version

1.0 of the SOAP/XML interface. An *i.*LON 100 using version 1.1 software will accept and respond to SOAP messages sent by applications written with version 1.0 of the SOAP/XML interface just as an *i.*LON 100 using version 1.0 software would.

1.4.1 Modified SOAP Applications and Functions

Table 1 lists the functions and properties that have been modified for version 1.1 of the SOAP/XML interface. Each function includes a brief explanation of how the function or property was modified for version 1.1. Detailed explanations of these modifications are included later in the document.

Table 1 *i.*LON 100 Version 1.1 SOAP/XML Interface Modifications

Function	**Description of Change**	**For More Information, See…**
AlarmNotifierRead	The AlarmNotifierRead function now returns two additional properties in its output: <UCPTlastEvent> and <UCPTtotalCount>. The <UCPTlastEvent> property indicates the last time an entry in the alarm log read by the function was modified or deleted. The <UCPTtotalCount> property indicates the total number of entries in the alarm log read by the function. In addition, the AlarmNotifierRead function now allows the user to read from the end of the alarm log backwards, if the <UCPTstop> property is specified and the <UCPTstart> property is left empty in the input supplied to the function. Further, when reading the Alarm History Log, the <UCPTstart> and <UCPTstop> input parameters now indicate a filter for log entry time. When reading the Alarm Summary Log, the <UCPTstart> and <UCPTstop> input parameters still indicate a filter for alarm time, as was the case in Version 1.0 of the SOAP/XML Interface. This change will affect the ordering of log entries returned when reading the Alarm History Log, because they will not be sorted by alarm time, as in version 1.0. Rather, they will appear in the order that each event was entered into the log.	*AlarmNotifierRead* on page 8-20

Function	Description of Change	For More Information, See…
AlarmGeneratorGet	The *i*.LON 100 has been modified to handle the properties defining the hysteresis levels and offset limits used by each Alarm Generator differently for certain data types. This only applies if you use the AlarmGeneratorGet function to return the configuration of an Alarm Generator with input data points that use the following format descriptions: SNVT_temp_f#US SNVT_temp_p#US SNVT_temp#US In this case, the values of the hysteresis level and offset limit properties returned by the function will be displayed using the SNVT_temp_f#US_diff format description. This rule applies to all formats that use the #US specifier.	*AlarmGeneratorGet* on page 7-5.
DataloggerGet	The *i*.LON 100 has been modified to handle the <UCPTlogMinDeltaValue> property, which defines the minuimum change in value that must occur for the data point update to be recorded by a Data Logger, differently for some data types. This only applies if you use the DataLoggerGet function to return the configuration of Data Logger with input data points that use the following format descriptions: SNVT_temp_f#US SNVT_temp_p#US SNVT_temp#US In this case, the <UCPTlogMinDeltaValue> property returned by the function will be displayed using the SNVT_temp_f#US_diff format type. This rule applies to all formats that use the #US specifier.	*DataLoggerGet* on page 6-6
DataLoggerRead	The DataLoggerRead function now two additional properties in its output: <UCPTlastEvent> and <UCPTtotalCount>. The <UCPTtotalCount> property indicates when the last time an entry in the data log read by the function was modified or deleted. The <UCPTtotalCount> property indicates the total number of entries in the data log read by the function. In addition, the DataLoggerRead function now allows the user to read from the end of the data log backwards, if the the <UCPTstop> property is specified and the <UCPTstart> property is left empty in the input supplied to the function.	*DataLoggerRead* on page 6-11

Function	**Description of Change**	**For More Information, See…**
DataServerRead	In version 1.0 of the *i.*LON 100 SOAP/XML interface, you could use the DataServerRead function to read the value of any data point, or group of data points. However, each data point had to be specified individually, by its name or index number. In version 1.1, you can specify a time stamp range in the input you supply to the DataServerRead function so that the i *i.*LON 100 only responds with data points whose last update occurred within the range. This allows your application to optimize throughput by only requesting data points whose value has changed since the last poll. The DataServerRead method also now accepts parameters to restrict the bus type (e.g. NVL or NVC) and total number of data points returned in the response message.	*DataServerRead* on page 5-19
<UCPTlifeTime>	The <UCPTlifeTime> property defines how old the value of a data point can be before the Data Server retrieves a new data value from the driver when an application requests its value. If the property is set to 0, the values of the data points will be copied from the *i.*LON 100 Data Server when an application requests them, and no update will be requested from the applicable driver. If this parameter is set to a positive value, the *i.*LON 100 Data Server will poll the driver for the current value of a data point each time an application requests it, and the time interval defined by the property has expired. The interval resets each time the value of a data point is retrieved. In version 1.0, you could set the <UCPTlifeTime> property by manually modifying it in the DP_NVL.XML or DP_NVE.XML configuration files. As of version 1.1, you can also temporarily override this value each time you call the DataServerRead function.	*DataServerRead* on page 5-19
DataServerWrite	You can use the DataServerWrite function to update the value of one or more data points. The DataServerWrite function now returns a fault response message if the *i.*LON 100 encounters errors updating any of the data points specified in the input to the function. The response message contains an element for each data point it was unable to update that describes why the data point could not be updated.	*DataServerWrite* on page 5-24

1.4.2 Changes to SOAP Message Formats

The SOAP body that must be included in every message sent to the *i.*LON 100, and in every output message returned by the *i.*LON 100, has been updated in version 1.1. The namespace URI has been modified to reflect the new version number of the *i.*LON 100.

In addition, the SOAP header attached to response messages sent by the *i.*LON 100 will include the time the message was sent, the IP address of the *i.*LON 100 that sent the message, and the port number used to send the transmission. These changes are described in Chapter 2 of this document.

2 SOAP Messages and the *i.*LON 100 WSDL File

This chapter contains general information about the SOAP/XML interface you will need to know before using the SOAP functions described in this manual. It includes the following major topics:

- *i.LON* 100 WSDL File. An overview of the *i.*LON 100 WSDL file, which defines the SOAP/XML interface. When writing applications to use the SOAP interface, some tools can import this file and automatically build a class structure for sending and receiving each message.
- Security. An overview of the security features provided by the *i.*LON 100 for SOAP applications.
- Formats of SOAP Messages. As described in Chapter 1, a SOAP message is sent to the *i.*LON 100 each time you invoke any of the functions described in this document. This section describes the formats that must be used for all SOAP messages that are sent to and from the *i.*LON 100.
- Writing SOAP Applications. SOAP applications for the *i.*LON 100 can be divided into two general groups: those that are written to configure the various applications of the *i.*LON 100, and those that are written to perform monitor and control tasks on the network that the *i.*LON 100 is attached to. This section sets a road map for you to follow as you learn how to write each kind of application.

2.1 *i.*LON *100 WSDL File*

Each *i.*LON 100 includes a WSDL (Web Service Description Language) file. This file defines the *i.*LON 100 SOAP/XML interface, and contains all the information an application will require to use the SOAP/XML interface. When writing applications to use the SOAP interface, some tools can import the WSDL file and automatically build a class structure for sending and receiving each message. The WSDL file is compatible with numerous programming development environments, such as Microsoft Visual Studio .NET ®.

See Chapter 14,

Using the SOAP Interface as a Web Service, for more detailed information the WSDL file. Chapter 14 contains step-by-step instructions you can follow when you reference the version 1.1 WSDL file with a Microsoft Visual Basic .NET project.

2.2 *Security*

You can add a basic level of security to the *i.*LON 100 SOAP/XML interface with the *i.*LON 100 Web Server Security and Parameters utility. Use this utility to add password protection to all web content served by the *i.*LON 100. Basic Access Authentication is the security mechanism used by the *i.*LON 100 web server for HTTP transactions. Basic Access Authentication is described by the IETF in RFC 2617:

http://www.ietf.org/rfc/rfc2617.txt

If you want all SOAP messages sent to your *i.*LON 100 to be authenticated, use the *i.*LON 100 Web Server Security and Parameters utility to password protect the *i.*LON 100 WSDL file at this path in the Web server: /WSDL/iLON100.WSDL.

A user name and password will then be required each time a SOAP message is sent to the *i.*LON 100. Since SOAP uses HTTP as a transport, you can use the user name and password pair for an entire HTTP session. As a result, you can use a single username and password to authenticate access to Web pages that send or receive multiple SOAP messages. If a SOAP message is sent to the *i.*LON 100 that does not contain the correct user name and password, it will be ignored. For instructions on using the *i.*LON Web Server Security and Parameters utility, see Chapter 13 of the *i.*LON *100 Internet Server User's Guide.*

To protect FTP access to the XML configuration files, the *i.*LON 100 requires a user name and password for every FTP session. This username and password default to "ilon", and can be re-defined with the *i.*LON 100 Security Web Page. The *i.*LON *100 Internet Server User's Guide* describes how to use this page.

2.3 Formats of SOAP Messages

As described in Chapter 1, a SOAP message is sent to the *i.*LON 100 each time you invoke a SOAP function. The content of the SOAP message, and the effect it has on the *i.*LON 100, is based on the input you supply to the function. The *i.*LON 100 reads the contents of the message, and adjusts its operating parameters of its applications accordingly. It also returns a response message describing the status or configuration of the modified application.

This section describes the basic format of the SOAP messages that are sent to the *i.*LON 100 when you invoke any of the functions described in this manual. It also describes the formats of the response SOAP messages that are returned by these functions.

NOTE: All SOAP messages sent to and from the *i.*LON 100 must adhere to the UTF-8 encoding standard. This is indicated by the <?xml version="1.0" encoding="utf-8" ?> tag included in each SOAP messages, as shown in the following sections. However, this restriction is not enforced by the *i.*LON 100 software. As a result, the *i.*LON 100 will accept SOAP messages that do not adhere to the UTF-8 encoding standard without throwing an error, which may result in invalid configuration data being written to your *i.*LON 100. To avoid this, you should program your application to ensure that all SOAP messages adhere to the UTF-8 encoding standard. For more information on the UTF-8 encoding standard, see http://www.ietf.org/rfc/rfc3629.txt.

2.3.1 Input Messages

The following represents the basic format of the SOAP messages that are sent to the *i.*LON 100 when you invoke any of the functions described in this manual. The sections following this sample describe each part of the SOAP message.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <echelon:FunctionName
   xmlns:echelon="http://wsdl.echelon.com/web_services_ns/ilon100/v1.1/message/">
         <Data> Data </Data>
      </echelon:FunctionName>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.3.1.1 SOAP Envelope

The SOAP envelope is the highest level of a SOAP message. The SOAP envelope for any message sent to the *i.*LON 100 must conform to the W3C SOAP 1.1 proposed Technical Recommendation:

http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

The SOAP envelope portion of the sample input message shown above includes the following:

```
<SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   ...
</SOAP-ENV:Envelope>
```

You will notice that the fourth line of this example includes the symbol "`...`" This is where the portion of the message known as the SOAP body would be. The SOAP body portion of the message is described in the next section.

2.3.1.2 SOAP Body

The SOAP body contains general information about the SOAP message, and contains the data that is passed to the function as input. The SOAP body conforms to the W3C SOAP 1.1 proposed Technical Recommendation mentioned above.

The SOAP header portion of the sample input message shown above includes the following:

```
<SOAP-ENV:Body>
      <echelon:FunctionName
xmlns:echelon="http://wsdl.echelon.com/web_services_ns/ilon100/v1.1/message/">
         <Data>Data</Data>
      </echelon:FunctionName>
   </SOAP-ENV:Body>
```

The name of the function that was invoked is passed as part of the SOAP body, and is prefixed by the string "Echelon." **In order to use the features included in version 1.1, the version1.1 namespace URI must be included as an attribute of the function name element**. The *i.*LON 100 namespace URI for version 1.1 of the SOAP interface is:

http://wsdl.echelon.com/web_services_ns/ilon100/v1.1/

By passing this Namespace URI in the input messages, the tool transmits version and platform compatibility information to the target server. In this way, a version 1.1 *i.*LON 100 could accept a version 1.0 SOAP message, and recognize from the namespace that it is a version 1.0 SOAP message. It would then process the message as though it were a version 1.0 *i.*LON 100. **As a result, all applications written for version 1.0 of the SOAP/XML interface are compatible with version 1.1 of the SOAP/XML interface.** The *i.*LON 100 will return the Namespace URI in all of the response messages it sends, so that a tool can use the Namespace identifier to check the version and platform of the SOAP interface.

2.3.1.2.1 <Data> Parameter

The <Data> parameter in the SOAP body is an essential part of the body of the SOAP message sent to the *i.*LON 100 when you call the functions described in this manual. The above example shows the string "**Data**" included in the <Data> parameter. **This represents the input that you must supply when you invoke any of the functions described in this manual.**

The majority of the functions of the SOAP/XML interface require a string of XML as input. When the function is called, the SOAP message is generated, the XML is inserted in the <Data> parameter, and the message is sent. The description of each SOAP function in this document includes a sample XML string you could supply as the <Data> parameter, and a description of how to build the XML string.

However, there are a few exceptions to this rule. **The DataPointRead, DataPointWrite, and DataPointResetPriority functions do not include a <Data> parameter as part of their input messgaes.** They take a series of input parameters, each of which contains a single value, as input. These input parameters are inserted in place of the <Data> parameter that is included in the SOAP body of all the other SOAP functions. This is described in more detail later in the document in Chapter 3, where the DataPoint functions are described.

2.3.2 Response Message

The following represents the basic format of the SOAP message returned by the *i.*LON 100 when you call any of the functions described in this document.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header>
       <p:messageProperties
       xmlns:p="http://wsdl.echelon.com/web_services_ns/ilon100/v1.1/type">
              <UCPTtimeStamp>2000-04-18T10:08:47.200-07:00</UCPTtimeStamp>
              <UCPTuniqueId>030000073782</UCPTuniqueId>
              <UCPTipAddress>172.25.137.100</UCPTipAddress>
              <UCPTport>80</UCPTport>
       </p:messageProperties>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
       <echelon:FunctionNameResponse
 xmlns:echelon="http://wsdl.echelon.com/web_services_ns/ilon100/v1.1/message/">
       <Result>Result</Result>
       </echelon:FunctionNameResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

There are two differences between the input messages sent to the *i.*LON 100, and those returned by the *i.*LON 100. The first is the inclusion of a SOAP header between the SOAP envelope and SOAP body. The second is the inclusion of the <Result> parameter contained within the SOAP body.

2.3.2.1 SOAP Header

The SOAP/XML interface has been modified in version 1.1 to include a SOAP header in all response messages. The SOAP header section of each response message is between the SOAP envelope and the SOAP body, and is shown below:

```
<SOAP-ENV:Header>
       <p:messageProperties
       xmlns:p="http://wsdl.echelon.com/web_services_ns/ilon100/v1.1/type">
              <UCPTtimeStamp>2000-04-18T10:08:47.200-07:00</UCPTtimeStamp>
              <UCPTuniqueId>030000073782</UCPTuniqueId>
              <UCPTipAddress>172.25.137.100</UCPTipAddress>
              <UCPTport>80</UCPTport>
       </p:messageProperties>
</SOAP-ENV:Header>
```

The SOAP Header contains general information about the message, and can be used to identify the *i.*LON 100 that sent it. This section is also tightly controlled by the W3C standards mentioned in the previous section. Every element in a SOAP Header, and all immediate child elements must be namespace qualified. Therefore, each user defined element contains a namespace prefix and a path to the unique Echelon Namespace.

The SOAP Header consists of a complex type with four fields describing different properties of the message:

- <UCPTtimeStamp> This field contains a time stamp indicating when the message was sent.
- <UCPTuniqueId> This field contains the Neuron ID of the *i*.LON 100, which is the third Neuron ID in the *i*.LON 100s block of addresses.
- <UCPTipAddress> This field contains the IP address of the *i*.LON 100 that sent the SOAP message.
- <UCPTport> This field contains the HTTP port specifier for the *i*.LON 100 web server.

2.3.2.2 <Result> Parameter

Another difference between response messages sent by the *i*.LON 100, and input messages sent to the *i*.LON 100, is the <Result> element contained within the SOAP body. The above example shows the string "**Result**" included in the <Result> parameter. **This represents the information that will be returned you invoke any of the functions described in this manual.**

The information that will be returned varies, depending on the function you are using. The description of each function included in this document includes a sample <Result> parameter that could be returned by the function, and information to help you interpret the contents of the <Result> parameter.

2.3.3 SOAP Error Responses

The following represents the basic format of the SOAP message returned by the *i*.LON 100 when the input you supply to the function causes an error.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <SOAP-ENV:faultcode>ErrorCode</SOAP-ENV:faultcode>
      <SOAP-ENV:faultstring>ErrorMessage</SOAP-ENV:faultstring>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

You will notice that the SOAP envelope of an error response contains an attribute called `<SOAP-ENV:faultcode>`**.** This contains the error code returned by the function. The next attribute is the <SOAP-ENV:faultstring> attribute**.** This represents the description of the error code returned by the function. The rest of the SOAP envelope and body is the same as those inlcuded in the input and response SOAP messages sent to and from the *i*.LON 100 (excluding the <Data> and <Result> parameters).

Table 2 lists the error codes and messages that the *i*.LON 100 SOAP interface may return.

Table 2 SOAP Error Codes

Error Code	Error Message

Error Code	Error Message
0	No Error
1	Unknown function call.
2	Parameter error. For example, the input you supplied to the function does not contain valid data, or no data was supplied to the function.
3	XML/Parser Error.
4	Tag missing.
5	Index missing
6	Index not found
7	Index invalid The index number you supplied to the function is greater than the maximum or less than the minimum allowed by the application. The allowable range of index numbers in the *i.*LON 100 is –32,768-32,767.
8	Can't create. This error may occur when you attempt to create a data point.
9	Can't delete. This error may occur when you attempt to delete a data point.
10	Can't set. This error may occur when attempting to modify the configuration of an existing item in the *i.*LON 100. For example, when attempting to write to the configuration of a data point.
11	Format Error
12	Command failed
13	The data point name referenced in the call to the function does not use the supplied index number.
14	Data point name not found in the *i.*LON 100 Data Server.
15	No Data
16	Field name not found. This will occur when attempting to read, write or set a data point that is structure, and you reference a structure field that does not exist.

2.4 Writing SOAP Applications

The majority of the SOAP functions and XML files described in this document will be useful to those who plan to use the SOAP/XML interface to configure the various applications of the *i.*LON 100. To begin learning about the SOAP functions and XML files you can use to do so, see Chapter 4, *i.LON 100 Applications and the SOAP/XML Interface*.

However, the SOAP/XML interface for the *i.*LON 100 also includes functions which allow you to write applications to monitor and control the data points you have created for your control network with LONMAKER and the *i.*LON 100 Configuration Software. Applications like these may be useful to anyone using the *i.*LON 100, including those who will only be using LONMAKER and the *i.*LON 100 Configuration Software to configure the *i.*LON 100's applications. For more information on these functions, see Chapter 3, *Monitoring and Controlling Data Points with the SOAP/XML Interface.*

3 Monitoring and Controlling Data Points with the SOAP/XML Interface

Any *i*.LON 100 user can make use of the SOAP/XML interface, even those who do not plan on using it to configure the applications of their *i*.LON 100. The SOAP interface includes functions that read and write the values of the data points defined on the *i*.LON 100. These functions do not affect the configuration of the *i*.LON 100 applications. You can use them to create applications to monitor and control data in the control network attached to your *i*.LON 100.

This chapter provides an overview of data points, and the functions you can use to read and write their values. It also describes the syntax of each of these functions, and a programming sample written in Microsoft Visual Basic .NET that may assist you when using these functions.

NOTE: You can use the functions described in this chapter to read or write to the value of a single data point at a time. This may create unnecessary network traffic if you use it to read or write to a large number of data points over a short period of time. The Data Server SOAP interface includes functions you can use to read or write to multiple data points at a time. These functions are described in Chapter 5, *Data Server*. However, you should review Chapter 4, *i.LON 100 Applications and the SOAP/XML Interface*, before attempting to use any of the functions described in Chapter 5.

3.1 Overview of Data Points

The *i*.LON 100 uses the concept of a data point to map logical names to *i*.LON 100 system variables, network variables defined on the *i*.LON 100 LonTalk interface, and explicitly addressed network variables. Data points provide the *i*.LON 100 applications and Web server a generic, open way to handle any piece of information from any type of network, such as the current value of a network variable in an LNS managed network, or a network variable on a self-installed node in a closed LONWORKS system.

This document describes how to use two kinds of data points:

- NVL data points for network variables that are local to the *i*.LON 100
- NVC data points for *i*.LON 100 system variables that maintain constant values

The *i*.LON 100 Data Server handles all the details of each data point that the various applications require, such as how often the value of a data point should be polled, its default value, its heartbeat, its current status, and its value.

At the Data Server layer, all data points have the same set of properties, regardless of which network or device they are local to. This made possible by the drivers that exist for each separate data point type, which handle all communication between the *i*.LON 100 Data Server and the network each data point is local to. Each driver on the *i*.LON 100 must be configured using a standard network management tool for that particular point type.

For example, you could use an LNS-based network management tool to configure the NVL points on the *i*.LON 100. The abstraction layer between the drivers and the Data Server provides a mechanism for all *i*.LON 100 applications to use data points of all types and from all devices in the same way.

One of the most important properties stored in the Data Server for each data point is the <UCPTvalue> property. The <UCPTvalue> property contains the current value of the data point. This property is updated by the Data Server in real time, and can be read or written to with the functions described in this chapter.

3.2 About This Chapter

This chapter describes the syntax of each of the functions you can use to read and write to the values of the data points in your network. It also includes code samples written in Microsoft Visual Basic .NET that may be useful when designing applications that use these functions.

NOTE: You can use the Data Point functions to read and write to the values of your data points. You must create data points and add them to the Data Server before using these functions. You can create NVL data points with LONMAKER. You can create NVC data points with the *i.*LON 100 Configuration Software, or with the SOAP interface.

For more information on the using the SOAP interface to create data points, see Chapter 5, *Data Server*. You should review Chapter 4, *i.LON 100 Applications and the SOAP/XML Interface*, before using the SOAP interface to create data points. For more information on the *i.*LON 100 Configuration Software, see the *i.*LON *100 Internet Server User's Guide*.

3.3 DataPointWrite

You can use the DataPointWrite function to write to the value of any data point in your network. You must reference the data point to be modified by the name (UCPTpointName) assigned to it in *i*.LON 100 Data Server in the input you supply to the function, as in the example below. The <UCPTpointName> of a data point is defined when it is created and added to the *i*.LON 100 Data Server. You can create data points with the *i*.LON 100 Configuration Software or with the DataServerSet function.

Chapter 5, *Data Server*, describes the DataServerSet function. However, you should review Chapter 4, *i.LON 100 Applications and the SOAP/XML Interface,* before attempting to use the function. For instructions on creating data points with the *i*.LON 100 Configuration Software, see the *i*.LON 100 *Internet Server User's Guide.*

If the specified data point is a structure, you can specify the field (UCPTfieldName) whose value should be written to by the function. Table 3 describes the rest of the parameters you will need to supply as input to this function.

You can only write to the value of one data point at a time with this function. However, you can use the DataServerWrite function to write to the values of multiple data points at once. Chapter 5, *Data Server*, describes the DataServerWrite function. However, you should review Chapter 4, *i.LON 100 Applications and the SOAP/XML Interface*, before attempting to use the function.

The sample below shows how the input parameters are filled into the SOAP message sent to the *i*.LON 100 when you call DataPointWrite. For programming samples writen in Microsoft Visual Basic .NET that call this function (and will fill in the SOAP message with the input data shown below), see *Using the DataPoint Functions With Visual Basic .NET* on page 3-11.

Input Parameters

```
<UCPTpointName>NVL_nvo01Switch</UCPTpointName>
<UCPTfieldName>value</UCPTfieldName>
<UCPTvalueDef>100.0</UCPTvalueDef>
<UCPTpriority>25</UCPTpriority>
<UCPTpropagate>1</UCPTpropagate>
```

Return Parameter

```
<Result>
  <UCPTpointName>NVL_nvo01Switch</UCPTpointName>
  <UCPTfieldName>value</UCPTfieldName>
</Result>
```

Table 3 describes the properties that you will need to supply as input when you call the DataPointWrite function. The function returns the name of the data point written to.

Table 3 DataPointWrite Input Properties

Property	Description

Property	Description
`<UCPTpointName>`	Enter the <UCPTpointName> of the data point to be written to. The name must begin with the following prefixes: • NVL_ for an NVL data point • NVC_ for an NVC data point **Note:** The names assigned to NVL data points follow the naming convention NVL_[NAME], where [NAME] represents that progammatic name assigned to the NV when it was created with LONMAKER. For example, if the progammatic name of the NV in LONMAKER is nvo01Switch_001, the <UCPTpointName> would be NVL_nvo01Switch. You can determine the progammatic name of an NV in LONMAKER by right-clicking it and selecting **Properties.**
`<UCPTfieldName>`	Field name. This can be left empty if the data point is not a structure, or if an entire structure is to be written to. **NOTE:** If this property is defined, the <UCPTvalueDef> property can only contain the actual value to be assigned to the data point, and not a pre-defined value definition.
`<UCPTvalueDef>`	Enter either the value definition (if the <UCPTfieldName> property is not filled in) or the actual value to be assigned to the data point. The format of the value entered must match the format type that the data point to be written to requires. The value definition is a string representing a preset value. This value must match the format type the data point requires. You can create value definitions for a data point using the *i.*LON 100 Configuration Software, or the DataServerSet SOAP function.
`<UCPTpriority>`	Enter the priority level to be assigned to the data point. The priority level must be specified as an integer between 0 (highest priority) and 255 (lowest priority). In order for your application to update the value of a data point successfully, the priority selected must be of equal or higher priority than that used by the last application to write to the data point. If no priority level is defined, the default value of 255 will be used. For a more detailed description of priority levels, see the next *Data Point Values and Priority Levels* on page 3-5.

Property	Description
`<UCPTpropagate>`	0 or 1. If you assign the default value 1 to the <UCPTpropagate> property, the change you make to the data point value will be propagated over the network. If you assign value 0 to this property, the change will be made in the *i.*LON 100 Data Server, but not over the network. This may be useful if you are writing to different fields of a structure with separate calls to DataPointWrite, and do not want to update the structure over the network until all of its fields have been written to.

3.3.1 Data Point Values and Priority Levels

As described in the previous section, the DataPointWrite function requires you to specify a priority level when writing to a data point. In order to successfully update the value of the data point, you must specify a higher priority level than the one used by the last application to write to the data point.

For example, consider a scenario where a SOAP application uses the DataPointWrite function to write to the value of a data point called NVL_nvoValue. Assume that the last application to write to the value of NVL_nvoValue used priority level 75 when it updated the data point. In that case, the current application must use a priority value between 0 and 75 to successfully write a new value to the data point.

Data point priority levels allow you to give some applications precedence over others when more than one application might attempt to update the same data point. Table 4 depicts a series of events where various applications write to the value of a single data point. For each event, the priority level used is listed, as well as a description of whether or not the update was successful, and why. This should help you understand how you can use data point priority levels to determine which applications will be given precedence when updating the value of a data point.

Table 4 Data Point Priority Levels and Values

Event	Priority Level Assigned	Result of Operation
Power-Up	255	The value of the data point is updated successfully.
Event Scheduler Updates Data Point	240	The value of the data point is updated successfully, as the priority used by the Event Scheduler is greater than that assigned to the data point during power-up.
Custom Application Invokes DataPointWrite	**75**	The value of the data point is updated successfully, as the priority used in the call to DataPointWrite is greater than that assigned to the data point by the Event Scheduler.
Event Scheduler Updates Data Point	240	The value of the data point is not updated successfully, as the priority used by the Event Scheduler is less than that used by the last application to update the data point.

Event	Priority Level Assigned	Result of Operation
Custom Application Invokes DataPointResetPriority	255	The custom application invokes the DataPointResetPriority function to reset the priority level assigned to the data point. This does not result in a change in the data point's value, but the priority level assigned to the data point is reset to 255, the lowest priority. At this point, all applications will be able to write to the data point. For more information on the DataPointResetPriority function, see *DataPointResetPriority* on page 3-10.
Event Scheduler Updates Data Point	240	The Event Scheduler successfully updates the value of the data point, as the priority level used here (240) is greater than that assigned to the data point by the DataPointResetPriority function.

3.4 DataPointRead

You can use the DataPointRead function to retrieve the current value assigned to a data point, as well as the values of several properties associated with the data point. You can use this function with any data point in your network.

You must reference the data point whose you value you want to read by its name (UCPTpointName) in the input you supply to the function, as in the example below. If the data point is a structure, you can specify the field whose value is to be retrieved by filling in the optional <UCPTfieldName> property.

You can only read the value of one data point at a time with this function. However, you can use the DataServerRead function to read the values of multiple data points at once. Chapter 5, *Data Server*, describes the DataServerRead function. However, you should review Chapter 4, *i.LON 100 Applications and the SOAP/XML Interface*, before attempting to use the function.

The sample below shows how the input parameters are filled into the SOAP message sent to the *i.*LON 100 when you call DataPointRead. For programming samples writen in Microsoft Visual Basic .NET that call this function (and will fill in the SOAP message with the input parameters shown below), see *Using the DataPoint Functions With Visual Basic .NET* on page 3-11.

Input Parameters

```
<UCPTpointName>NVL_nvo03Switch</UCPTpointName>
<UCPTfieldName>state</UCPTfieldName>
```

Return Parameters

```
<Result>
  <UCPTpointName>NVL_nvo03Switch</UCPTpointName>
  <UCPTfieldName>state</UCPTfieldName>
  <UCPTlocation>MainBuilding\First Floor\Meetingroom\Light<UCPTlocation>
  <UCPTpointUpdateTime>2001-07-24T01:47:22.000+03:00</UCPTpointUpdateTime>
  <UCPTvalue>1</UCPTvalue>
  <UCPTvalueDef></UCPTvalueDef>
  <UCPTunit></UCPTunit>
  <UCPTpointStatus>AL_NO_CONDITION</UCPTpointStatus>
  <UCPTpriority>250</UCPTpriority>
  <UCPTformatDescription>SNVT_switch<UCPTformatDescription>
  <UCPTbaseType>BT_STRUCT</UCPTbaseType>
</Result>
```

Table 5 describes the properties that the DataPointRead function returns in the <Result> parameter. Note that the <UCPTpointName> and <UCPTfieldName> properties are also supplied as input in the <Data> parameter.

Table 5 DataPointRead Output Properties

Property	Description

Property	Description
`<UCPTpointName>`	Enter the <UCPTpointName> of the data point to be written to. The name must begin with the following prefixes: • NVL_ for an NVL data point • NVC_ for an NVC data point **Note:** The names assigned to NVL data points follow the naming convention NVL_[NAME], where [NAME] represents that progammatic name assigned to the NV when it was created with LONMAKER. For example, if the progammatic name of the NV in LONMAKER is nvo01Switch_001, the <UCPTpointName> would be NVL_nvo01Switch. You can determine the progammatic name of an NV in LONMAKER by right-clicking it and selecting **Properties.**
`<UCPTfieldName>`	The name of the data point field whose value is to be read. This property should be left empty if the data point is not a structure. If defined, the <UCPTvalueDef> property will not be included in the <Result> parameter.
`<UCPTlocation>`	An alphanumeric string of up to 128 characters that describes the location of the data point. This user-defined property is established when the data point is created. You could use it to organize your data points by physical location or device.
`<UCPTpointUpdateTime>`	A timestamp indicating the last time the value of the data point was updated. This timestamp is expressed in local time, with an appended time zone indicator that indicates the difference between local time and Coordinated Universal Time (UTC). UTC is the current term for what was commonly referred to as Greenwich Meridian Time (GMT). Zero (0) hours UTC is midnight in Greenwich England, which lies on the zero longitudinal meridian. Universal time is based on a 24 hour clock, therefore, an afternoon hour such as 4 pm UTC would be expressed as 16:00 UTC. The timestamp uses the following format: [YYYY-MM-DD]T[HH:MM:SS.MSS]+/-[HH:MM] The first segment of the timestamp [YYYY-MM-DD] represents the date. The second segment (T[HH:MM:SS.MSS]) of the timestamp represents the local time, expressed in hours, minutes, seconds and milliseconds. The third segment (+/-[HH:MM]) represents the difference between the local time listed in the second segment and UTC. This segment begins with a + or a -. The + indicates that the local time is ahead of UTC, the - indicates the local time is behind UTC. Consider the following example: 2002-08-13T10:24:37.111+02:00 This timestamp indicates a local date and time of 10:24 AM and 37.111 seconds, on August 13, 2002. Because the third part of the segment reads +02:00, we know the local time here is 2 hours ahead of UTC.
`<UCPTvalue>`	The value currently assigned to the data point.

Property	Description
`<UCPTvalueDef>`	The value definition being used by the data point. If the data point is not using a value definition, or if you defined the <UCPTfieldName> property in the input you supplied to the function, this field will appear blank. Value definitions are user-defined strings that represent preset values the data point can be updated to. For example, the value assigned to a value definition called ON for a SNVT_switch data point might be "100.0 1." You can create value definitions for a data point with the DataServerSet function, or with the i.LON 100 Configuration Software.
`<UCPTunit>`	Unit type of the data point. This property is configured based on the network variable type of the data point.
`<UCPTpointStatus>`	The current status of the data point. This property is updated in real time by the Data Server, and can be used when setting up Alarm Generators and Alarm Notifiers with the i.LON 100. You can create Alarm Generators and Alarm Notifiers with the i.LON 100 Configuration Software, or with the SOAP interface. For information on using the SOAP interface to create Alarm Generators and Alarm Notifiers, see Chapters 7 and 8 of this document.
`<UCPTpriority>`	The priority level currently assigned to the data point. You can set the priority level of a data point with the DataPointWrite or DataPointResetPriority functions. For more information on priority levels, see *Data Point Values and Priority Levels* on page 3-5.
`<UCPTformatDescription>`	The format description of the data point. This determines many factors about the data point, including the type of values it takes, and its base type. This could be any standard (SNVT) format type included in the i.LON 100 resource files, or any user-defined (UNVT) format type included in resource files uploaded to the i.LON 100. For more information on the i.LON 100 resource files, see *i.LON 100 Resource Files* on page 4-8. The SNVT format types included in the i.LON 100 resource files are also listed and described in the SNVT Master List, which can be downloaded as a PDF by selecting the Documentation link on Echelon's Support Web site: http://www.echelon.com/support
`<UCPTbaseType>`	This read-only property is assigned to the data point automatically based on the data point's <UCPTformatDescription>. It defines the base type of the data point, as defined in the base_type_t enumeration in the BAS_Controller resource files for the i.LON 100. For more information on the i.LON 100 resource files, see *i.LON 100 Resource Files* on page 4-8.

3.5 DataPointResetPriority

You can use the DataPointResetPriority function to reset the priority level of a data point to 255, the lowest priority level. This would allow any application to write to the value of the data point.

You must reference the data point to be affected by its <UCPTpointName> in the input you supply to the function, as in the example below. In addition, you must specify a priority level of equal or higher priority than that currently assigned to the data point in the input you supply to the function. Only then will the priority level assigned to the data point be reset.

For example, if the priority level currently assigned to a data point is 30, then priority level specified in the input must be between 0 and 30. For a more detailed description of data point priority levels, see *Data Point Values and Priority Levels* on page 3-5.

NOTE: You can only reset the priority of one data point at a time with this function. However, you can use the DataServerResetPriority function to read the values of multiple data points at once. Chapter 5, *Data Server*, describes the DataServerResetPriority function. However, you should review Chapter 4*, i.LON 100 Applications and the SOAP/XML Interface*, before attempting to use the function.

The sample below shows how the input parameters are filled into the SOAP message sent to the *i.*LON 100 when you call DataPointResetPriority. For programming samples written in Microsoft Visual Basic .NET that call this function, and will fill in the SOAP message with the input data shown below, see *Using the DataPoint Functions With Visual Basic .NET* on page 3-11.

Input Parameters	`<UCPTpointName>`**`NVL_nvo01Switch`**`</UCPTpointName>`
	`<UCPTpriority>`**`25<`**`/UCPTpriority>`

Return Parameters	`<Result>`**`NVL_nvo01Switch`**`<Result>`

3.6 Using the DataPoint Functions With Visual Basic .NET

You can call the DataPoint functions from development environments such as Microsoft Visual Basic .NET by referencing the *i.*LON 100 WSDL file. Chapter 14*, Using the SOAP Interface as a Web Service*, describes how to reference the *i.*LON 100 WSDL file in Microsoft Visual Basic .NET.

This chapter describes the syntax you will use when you call each of these functions, and provides a code sample written in Visual Basic .NET that invokes each of them.

3.6.1 DataPointWrite

Syntax: `return=ilonWebReference.DataPointWrite(`*`UCPTpointName`*` As String, `*`UCPTfieldName`*` As String, `*`UCPTvalue`*` As String, `*`UCPTpropagate`*` As Integer, `*`UCPTpriority`*` As Integer)`

Element	Description
return	DataPointWriteReturnType variable
ilonWebReference	Instance of the *i.*LON 100 WSDL file Web service.
Input Parameters	For descriptions of the individual input parameters that you must supply to the function, see *DataPointWrite* on page 3-3.

Returns: The function returns an object containing two properties: the name of the data point written to (UCPTpointName), and the name of the field written to (UCPTfieldName).

3.6.2 DataPointRead

Syntax: `return=ilonWebReference.DataPointRead(`*`UCPTpointName`*` As String, `*`UCPTfieldName`*` As String)`

Element	Description
return	DataPointReadReturnType variable
ilonWebReference	Instance of the *i.*LON 100 WSDL file Web service.
Input Parameters	For descriptions of the input parameters that you must supply to the function, see *DataPointRead* on page 3-7.

Returns: The function returns an object containing each of the properties described in Table 5 on page 3-7. Each property can be retrieved from this object individually by referencing the property name listed in Table 5, as shown in the programming sample later in this section.

3.6.3 DataPointResetPriority

Syntax: `return=ilonWebReference.DataPointResetPriority(`*`UCPTpointName`*` As String, `*`UCPTpriority`*` As Integer)`

Element	Description
return	String variable
ilonWebReference	Instance of the *i.*LON 100 WSDL file Web service.

Input Parameters	For descriptions of the input parameters that you must supply to the function, see *DataPointResetPriority* on page 3-10.

Returns: The function returns a string containing the name of the data point reset by the function.

3.6.4 Programming Samples

The following programming sample, written in Microsoft Visual Basic .NET, invokes all three of the DataPoint functions. It uses the DataPointWrite function to write to the value of a data point called NVL_nvo01Switch. It then uses DataPointResetPriority to reset the priority level assigned to the data point, so that other applications can write to its value later.

Once the data point has been updated, the DataPointRead function is called. Each property included in the object returned by the function is then extracted and displayed in a text box.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e _
        As System.EventArgs) Handles Button1.Click

    'Create an instance of the i.LON 100 Web service, and return
    'objects for the three functions to be called. The WSDL file
    'defines a return type for DataPointRead and DataPointWrite. The
    'DataPointResetPriority returns a string containing the name of the
    'data point reset by the function.

    Dim ilon As New WebReference1.iLON100()
    Dim retRead As New WebReference1.DataPointReadReturnType()
    Dim retWrite As New WebReference1.DataPointWriteReturnType()
    Dim retPriority As String

    'Create variables to store the input for each function.
    Dim S As String
    Dim pointName As String
    Dim fieldName As String
    Dim value As String
    Dim propagate As String
    Dim priority As String

    'Specify the IP location of the i.LON 100 by replacing the default
    ' "localhost" with the IP address of the i.LON 100.

    S = ilon.Url
    ilon.Url = S.Replace("localhost", "10.5.0.50")

    'Define the data point to be written to, and the value to assign
    'it. In this case, the field name is left blank, as the data point
    'the program is writing to is not a structure.

    pointName = " NVL_nvo01Switch "
    fieldName = ""
    value = "0.0 0"
    propagate = "1"
    priority = "50"
    'Call DataPointWrite, and then call DataPointResetPriority. Note
    'that the priority 'level used in the call to
    'DataPointResetPriority is less than that used in the call to
    'DataPointWrite. This resets the priority assigned to
    'NVL_nvo01Switch to 255, and allows all other applications to write
    'to the data point. The propagate value used is in each call is 1,
    'so that the change is updated over the network.

    retWrite = ilon.DataPointWrite(pointName, fieldName, value, _
            priority, propagate)
    priority = "45"
    ilon.DataPointResetPriority(pointName, priority)

    'Call DataPointRead and display the properties contained in the
    'object returned by the function in text boxes. Note that the names
    'used to extract each property from the return object match the
    'property names used in the sample SOAP parameters earlier in this
    'chapter.

    retRead = ilon.DataPointRead(pointName, fieldName)
    TextBox1.Text = retRead.UCPTbaseType
    TextBox2.Text = retRead.UCPTfieldName
    TextBox3.Text = retRead.UCPTformatDescription
    TextBox4.Text = retRead.UCPTlocation
    TextBox5.Text = retRead.UCPTpointName
    TextBox6.Text = retRead.UCPTpointStatus
    TextBox7.Text = retRead.UCPTpointUpdateTime
    TextBox8.Text = retRead.UCPTpriority
    TextBox9.Text = retRead.UCPTunit
    TextBox10.Text = retRead.UCPTvalue
    TextBox11.Text = retRead.UCPTvalueDef
End Sub
```

4 *i.*LON 100 Applications and the SOAP/XML Interface

This chapter provides an overview of the applications supported by the *i.*LON 100, and of how you can use the SOAP/XML interface to configure these applications and use the data they generate. This chapter includes the following major sections:

- Overview of i.*LON* 100 Applications. This section provides a description of each of the applications that the *i.*LON 100 supports.
- i.LON 100 XML Configuration Files. The configuration of each *i.*LON 100 application is stored in an XML file. This section lists those XML files, and indicates where they are stored on the *i.*LON 100.
- i.LON 100 SOAP Functions. Each *i.*LON 100 application includes a set of SOAP functions that can be used to configure that application. This section lists and describes the functions provided for each application, and references where each function is described in more detail in this document. It also provides information you will require when constructing the input to be supplied to each function.
- i.LON 100 Resource Files. The *i.*LON 100 resource files contain information you will need when using the SOAP functions. This section describes how to use the resource files.
- List, Get, Set and Delete Functions. You will notice when reviewing the *i.LON 100 SOAP Functions* section that each application has separate List, Get, Set and Delete functions. Together, these functions form a symmetric interface, and you may find this symmetry very useful when programming your SOAP applications. This section describes how you might do so.
- Performance Issues. This section lists performance issues you should consider when using the SOAP/XML interface.
- Getting Started. This section provides a roadmap to follow when configuring the *i.*LON 100 applications. The most important part of this roadmap is that you must configure the *i.*LON 100 Data Server before configuring any other applications.

*4.1 Overview of i.*LON *100 Applications*

You need to build the *i.*LON 100 Data Server and create the data points you need to manage your control network before using the SOAP/XML interface to configure your applications. Chapters 5 of this document describes data points, and how to build the *i.*LON 100 Data Server. Once you have built the Data Server, you can use the SOAP/XML interface to configure the following *i.*LON 100 applications:

- Data Logging – You can configure the *i.*LON 100 to record updates to the data points on your network by creating Data Loggers. Each Data Logger will have its own log file, which will contain an entries for each of the updates to the data points it is monitoring. These logs can be downloaded and read using the Internet File Transfer Protocol (FTP), or retrieved using a SOAP function called DataLoggerRead. Table 6 provides a brief description of DataLoggerRead and the other functions you can use to create and manage your Data Loggers. These functions are described in Chapter 6 of this document.

- Alarming – You can configure the *i.*LON 100 to trigger alarms based on the values and statuses of the data points in your control network. The *i.*LON 100 can be configured to update any data point in the LONWORKS network, log the conditions to one or more data logs, or send out emails notifying recipients of the alarms and the conditions that triggered them. Alarms can be configured to shut off automatically when certain conditions are met, or they can be configured to require manual clearance. You will create Alarm Generators and Alarm Notifiers to manage these alarming tasks. Table 6 provides a brief description of the functions you can use to do so. These functions are described in detail in Chapters 7 and 8 of this document.

- Analog Function Blocks – You can use the Analog Function Block application to perform statistical operations on the values of any of the data points in your network. Table 6 provides a brief description of the functions you can use to do so. These functions are described in detail in Chapter 9 of this document.
- Scheduling – The *i.*LON 100 can be used to create daily and weekly schedules, as well as exception schedules and override schedules. These schedules can drive the inputs to data points bound to any LONWORKS device. You can create Event Schedulers and Event Calendars to manage these tasks. Table 6 provides a brief description of the functions you can use to do so. These functions are described in detail in Chapters 10 and 11 of this document.

- Type Translation – You can use the Type Translator application to translate data from one network variable data type to another. You will need to create Type Translators, and optionally Type Translator Rules, to translate your data. Table 6 provides a brief description of the functions you can use to do so. These functions are described in detail in Chapters 12 and 13 of this document.

4.2 *i.*LON *100 XML Configuration Files*

As described in Chapter 1, the configurations of each *i.*LON 100 application is stored in an XML file. The *i.*LON 100 contains the following configuration files:

/root/config/software/AlarmGenerator.XML
/root/config/software/AlarmNotifier.XML
/root/config/software/AnalogFB.XML
/root/config/software/EventCalendar.XML
/root/config/software/EventScheduler.XML
/root/config/software/DataLogger.XML
/root/config/software/TypeTranslator.XML
/root/config/software/DataServer/DP_NVL.XML
/root/config/software/DataServer/DP_NVC.XML

NOTE: The /root/config/software directory includes a sub-directory called TranslatorRules, which contains several XML files you can use when configuring your Type Translators. It also contains a file called RNI.XML, which contains configuration data used by the *i.*LON 100 remote network interface (RNI). There is no SOAP interface for the RNI application, and you should not manually edit the RNI.XML file. You can configure the RNI application using the *i.*LON 100 configuration web pages. For more information on this, see the *i.*LON *100 Internet Server User's Guide.*

Each application inludes a Set function. You can use the Set function to create and write to the applicable XML file. The *i.*LON 100 will modify the XML file, and the operating parameters of the associated application, each time it receives a Set message. The next section, *i.LON 100 SOAP Functions,* lists the other functions that can be used with each application.

As an alternative to using SOAP, you can modify the files manually using an ASCII-text or XML editor, and then download them to the *i.*LON 100 via FTP. Echelon does not recommend this, as you will need to reboot the *i.*LON 100 for it to read the downloaded files, and the *i.*LON 100 will not perform error-checking on the downloaded XML files.

Chapters 5-13 of this document describe the content of each of the *i.*LON 100 XML configuration files, the applications they support, and the SOAP functions you can use to manage them in detail. **Review the rest of this chapter before proceeding to any of Chapters 5-13, as it provides background information you will need when you use the SOAP/XML interface.**

4.3 *i.*LON *100 SOAP Functions*

Each of the XML files listed in the previous section will contain elements and properties that define the configuration of an *i.*LON 100 application, and the configuration of the items or instances that have been added to that application. For example, the AlarmGenerator.XML file defines the global configuration properties associated with the Alarm Generator application, as well as the configuration of each Alarm Generator that you have added to the *i.*LON 100.

Table 6 provides an overview of the the functions you can use to write to these XML files, build the *i.*LON 100 Data Server, configure the applications of your *i.*LON 100, and read the data that your applications generate. **It is highly critical that you review the rest of this chapter before using these functions.** This chapter provides background information on the SOAP interface you will need when reading the rest of this document.

Table 6 *i.*LON 100 SOAP Functions

Function Names	Description
DataServerList DataServerGet DataServerSet DataServerDelete DataServerRead DataServerWrite DataServerResetPriority	Use the DataServerList function to return the index number, name, and location of each data point that you have added to the *i.*LON 100 Data Server. You can use the DataServerGet function to return the configuration of any of these data points. Use the DataServerSet function to add data points to the *i.*LON 100 Data Server, or to update the configuration of the data points that are already in the Data Server. Use the DataServerRead and DataServerWrite functions to read and write to the current values of any of the data points in the network. Use the DataPointResetPriority function to reset the priority of any of these data points. Use the DataServerDelete function to delete any data point. For more information on these functions, see *Data Server* on page 5-1.

Function Names	Description
DataloggerList DataloggerGet DataloggerSet DataLoggerRead DataLoggerClear DataloggerDelete	Use the DataLoggerList function to return the index number, last update time, description, and functional block name of each Data Logger that you have added to the *i*.LON 100. You can use the DataLoggerGet function to return the configuration of any of these Data Loggers. Use the DataLoggerSet function to add new Data Loggers to the *i*.LON 100, or to overwrite the configuration of existing Data Loggers. Use the DataLoggerDelete function to remove Data Loggers from the *i*.LON 100. Use the DataLoggerRead function to read data from the log files generated by your Data Loggers. Use the DataLoggerClear function to remove data from the log files. For more information on these functions, see *Data Loggers* on page 6-1.
AlarmGeneratorList AlarmGeneratorGet AlarmGeneratorSet AlarmGeneratorDelete	Use the AlarmGeneratorList function to return the index number, last update time, description, and functional block name of each Alarm Generator that you have added to the *i*.LON 100. You can use the AlarmGeneratorGet function to return the configuration of any of these Alarm Generators. Use the AlarmGeneratorSet function to add new Alarm Generators to the *i*.LON 100, or to overwrite the configuration of existing Alarm Generators. Use the AlarmGeneratorDelete function to remove Alarm Generators from the *i*.LON 100. For more information on these functions, see *Alarm Generator* on page 7-1.
AlarmNotifierList AlarmNotifierGet AlarmNotifierSet AlarmNotifierDelete AlarmNotifierRead AlarmNotifierWrite AlarmNotifierClear	Use the AlarmNotifierList function to return the index number, last update time, description, and functional block name of each Alarm Notifier that you have added to the *i*.LON 100. You can use the AlarmNotifierGet function to return the configuration of any of these Alarm Notifiers. Use the AlarmNotifierSet function to add new Alarm Notifiers to the *i*.LON 100, or to overwrite the configuration of existing Alarm Notifiers. Use the AlarmNotifierDelete function to remove Alarm Notifiers from the *i*.LON 100. Use the AlarmNotifierRead function to read the log files generated by your Alarm Notifiers. Use the AlarmNotifierWrite function to comment on and acknowledge the entries in the log files. Use the Alarm NotifierClear function to remove entries from the log files. For more information on these functions, see *Alarm Notifier* on page 8-1.
AnalogFBList AnalogFBGet AnalogFBSet AnalogFBDelete	Use the AnalogFBList function to return the index number, last update time, description, and functional block name of each Analog Function Block that you have added to the *i*.LON 100. You can use the AnalogFBGet function to return the configuration of any of these Analog Function Blocks. Use the AnalogFBSet function to add new Analog Function Blocks to the *i*.LON 100, or to overwrite the configuration of existing Analog Function Blocks. Use the AnalogFBDelete function to remove Analog Function Blocks from the *i*.LON 100. For more information on these functions, see *Analog Function Block* on page 9-1

Function Names	Description
EventSchedulerList EventSchedulerGet EventSchedulerSet EventSchedulerDelete	Use the EventSchedulerList function to return the index number, last update time, description, and functional block name of each Event Scheduler that you have added to the *i*.LON 100. You can use the EventSchedulerGet function to return the configuration of any of these Event Schedulers. Use the EventSchedulerSet function to add new Event Schedulers to the *i*.LON 100, or to overwrite the configuration of existing Event Schedulers. Use the EventSchedulerDelete function to remove Event Schedulers from the *i*.LON 100. For more information on these functions, see *Event Scheduler* on page 10-1.
EventCalendarList EventCalendarGet EventCalendarSet EventCalendarDelete	Use the EventCalendarList function to return the index number, last update time, description, and functional block name of each Event Calendar that you have added to the *i*.LON 100. You can use the EventCalendarGet function to return the configuration of any of these Event Calendars. Use the EventCalendarSet function to add new Event Calendars to the *i*.LON 100, or to overwrite the configuration of existing Event Calendars. Use the EventCalendarDelete function to remove Event Calendars from the *i*.LON 100. For more information on these functions, see *Event Calendar* on page 11-1.
TypeTranslatorList TypeTranslatorGet TypeTranslatorSet TypeTranslatorDelete	Use the TypeTranslatorList function to return the index number, last update time, description, and functional block name of each Type Translator that you have added to the *i*.LON 100. You can use the TypeTranslatorGet function to return the configuration of any of these Type Translators. Use the TypeTranslatorSet function to add new Type Translators to the *i*.LON 100, or to overwrite the configuration of existing Type Translators. Use the TypeTranslatorDelete function to remove Type Translators from the *i*.LON 100. For more information on these functions, see *Type Translator* on page 12-1.
TypeTranslatorRuleList TypeTranslatorRuleGet TypeTranslatorRuleSet TypeTranslatorRuleDelete	Use the TypeTranslatorRuleList function to return the index number, last update time, description, and functional block name of each Type Translator Rule that you have added to the *i*.LON 100. You can use the TypeTranslatorRuleGet function to return the configuration of any of these Type Translator Rules. Use the TypeTranslatorRuleSet function to add new Type Translator Rules to the *i*.LON 100, or to overwrite the configuration of existing Type Translator Rules. Use the TypeTranslatorRuleDelete function to remove Type Translator Rules from the *i*.LON 100. For more information on these functions, see *Type Translator Rules* on page 13-1.

4.3.1 <Data> Parameter

As described in Chapters 1 and 2 of this document, a SOAP message is sent to the *i*.LON 100 each time you invoke any of the functions listed in Table 6. The *i*.LON 100 reads the message, and adjusts its configuration or operating parameters based on its contents. An essential part of each SOAP message sent to the *i*.LON 100 is the <Data> parameter. The <Data> parameter contains the input that you must supply to the SOAP functions when you invoke them.

This input is a string of encoded XML containing a list of objects, or the desired settings of any number of the configuration properties associated with the *i.*LON 100. The contents of the string vary by function. **The string of encoded XML contained within the <Data> parameter is the single parameter you will supply as input when calling the SOAP functions described in Chapters 5-13 of this document.** This differs from the DataPoint functions described in Chapter 3, as those functions take a series of parameters as input, instead of the singe <Data> parameter.

NOTE: Some programming environments will require you to include the <Data> parent element in the input you supply to each function. Others, such as Microsoft Visual Studio .NET, do not. They insert the <Data> parent element into the SOAP message automatically when the function is invoked, and as a result you are only required to supply the contents of the <Data> parameter (i.e. the XML string only) as input.

The properties and attributes that you must define in the XML string passed within the <Data> parameter when calling each SOAP function are described in Chapters 5-13 of this document. In addition, the *i.*LON 100 resource files contain format definitions and descriptions of all configuration properties and network variable used on the *i.*LON 100. This information will be necessary when determining the values to assign to each property within the XML string. For more information on the *i.*LON 100 resource files, see the next section, *i.LON 100 Resource Files*.

This document includes a sample <Data> parameter with sample values that you could supply to each function included in the SOAP interface. For ease of understanding, these samples are shown in standard XML format. **However, all data contained within the <Data> parameter must be passed over the network in encoded XML format.**

Some programming environments, such as Microsoft Visual Studio .NET, will accept the data to be included in the <Data> parameter in standard XML format, and convert it to encoded XML format before the SOAP message is sent to the *i.*LON 100. In this case, you can enter the data in standard XML format when coding your application.

Other programming environments will not automatically convert the data from standard XML to encoded XML. This requires you to supply the string to the function in encoded XML format when you program your application. The following sections describe the difference between encoded XML and standard XML.

NOTE: The input supplied to the DataPointRead, DataPointWrite, and DataPointResetPriority functions requires a different format. For more information, see *Formats of SOAP Messages* on page 2-2.

4.3.1.1 Encoded XML and Standard XML

The following sample shows a SOAP message in enocded XML format, as it would appear when transmitted over the network to the *i.*LON 100. The string inside the <Data> parameter is a valid XML structure in the encoded format discussed in the previous section. This prevents the encapsulated XML data from being interpreted as individual parameters in the SOAP message.

The encoding used in the *i.*LON 100 SOAP interface uses character escaping for the &, < and > characters as defined for XML 1.0 by the W3C in "Extensible Markup Language (XML) 1.0" (http://www.w3.org/TR/2000/REC-xml-20001006). The rules for this character escaping are as follows:

- **<** character is replaced by **&lt;**
- > character is replaced by **&gt;**

- & character is replaced by **&amp;**

For both the <Data> and <Result> parameters (see next section), the SOAP application will handle the value as a string. This string can be directly decoded and manipulated by an XML parsing engine. All input strings passed over the network in the <Data> parameter must be valid, encoded XML, and all strings returned by the *i.*LON 100 within the <Result> parameter will be passed over the network as valid, encoded XML.

The following represents a sample <Data> parameter as it would appear when passed over the network. The contents of the <Data> parameter here are in encoded XML format:

```
<Data>
   &lt;iLONApplication&gt;
     &lt;UCPTindexValue&gt;3&lt;/UCPTindexValue&gt;
     &lt;UCPTpointName&gt;NVL_nvo03Switch&lt;/UCPTpointName&gt;
   &lt;/iLONApplication&gt;
</Data>
```

To make this message format more readable, the above example would appear in standard XML format in this document, as shown below. The sample below, and all of the sample messages in this manual for the Data Server, Data Logger, Alarm Generator, Alarm Notifier, Event Calendar, Event Scheduler, or Type Translator applications, are not written as the messages actually appear when transmitted over the network. It is important to note this when using the sample XML strings provided in this document.

```
<Data>
   <iLONApplication>
      <UCPTindex>3</UCPTindex>
      <UCPTpointName>NVL_nvo03Switch</UCPTpointName>
   </iLONApplication>
</Data>
```

*4.4 i.*LON *100 Resource Files*

There are many configuration properties you can configure using the SOAP functions described in this document. This document provides a general description of each property, and other information you will need when configuring each one, such as minimum and maximum values for scalar properties, and maximum string lengths for string properties. This information is also contained in the *i.*LON 100 resource files. In order to successfully send a SOAP message to the *i.*LON 100, all data in the message must be formatted as described in this document, and in the resource files.

The *i.*LON 100 resource files are added to the LNS resource file catalog by the *i.*LON 100 Configuration Software installation utility, but they also exist locally on the *i.*LON 100. In fact, like LNS, the *i.*LON 100 maintains a catalog of resource files to use when formatting data in SOAP messages, network variable updates, and web tag data from the *i.*LON 100 web server.

You can use the Node Builder Resource Editor, which is included on the *i.*LON 100 Installation CD, to create new resource files for your own custom data point types and formats. Note that when creating custom resource files on a PC, it is common to organize the files into subdirectories such as:

```
C:\LonWorks\Types\User\MyCompany\MyResourceFiles.*
```

However, when adding these files to the *i.*LON you must FTP them to the following location:

```
/root/lonworks/types/MyResourceFiles.*
```

You only need to FTP your own custom resource files to the *i.*LON. If the name of your file set is "MyResourceFiles", then you must copy every file which starts with the name "MyResourceFiles". After you have copied these files to the *i.*LON you must reboot to be able to use the new type definitions and formats. During boot the *i.*LON reads the resource files in this directory and updates its local catalog accordingly.

4.4.1 LonMark Standard Network Variable Type (SNVT) Device Resource Files

SNVT device resource files describe the data structures within LonMark SNVTs, and the formats used to display SNVT data. On the *i.*LON 100, you can find these files in the directory `/root/lonworks/types.` They are named `STANDARD.ENU`, `STANDARD.TYP`, `STANDARD.FMT`, and `STANDARD.FPT`.

The default format for a SNVT is its native format, as described in `STANDARD.FMT.` When you add a data point to the *i.*LON 100, you will assign that data point a fotmat type. If a specific SNVT format is desired for a particular data point, the <UCPTformatDescription> of that data point must be set the name of that SNVT format. For example:

```
    <UCPTformatDescription>SNVT_temp_f</UCPTformatDescription>
```

The <UCPTformatDescription> property is described in more detail in Chapter 5, *Data Server*. You can browse the entire SNVT device resource files online at http://types.lonmark.org.

4.4.2 Standard Configuration Property Type (SCPT) Device Resource Files

This is a set of files that describes the data structures within SCPTs, and also describes the formats used to display SCPT data. On the *i.*LON 100, these files can be found in the directory /root/lonworks/types, and are named STANDARD.ENU, STANDARD.TYP, STANDARD.FMT and STANDARD.FPT.

Many configuration properties that are used by the *i.*LON 100 applications are based on the SCPTs defined in these files. The information provided in this document, and in the SCPT resource files, will help you determine what values to assign to the SCPTs referenced by the *i.*LON 100.

You can browse the entire SCPT device resource files online at http://types.lonmark.org.

4.4.3 User Network Variable Type (UNVT) Device Resource Files

Device manufacturers create UNVT device resource files to describe non-standard, manufacturer specific network variables. Using the same mechanisms as the standard resource files, these files describe how to format data from a particular manufacturer's device. On the *i.*LON 100, you can find all device resource files in the directory `/root/lonworks/types`.

To specify UNVT formats a fully qualified format name is required as follows:

```
#<progID>[<selector>].<format name>
```

In this syntax, the "#", "[", "]" and "." characters are literal characters. A hex byte string (in the "`RAW_HEX_PACKED`" format described below) represents the program ID. The selector is a one-digit string. It represents a filter that indicates relevant parts of the program ID, and may be one of the following:

0 - Standard

1 - Device Class

2 - Device Class and Usage

3 - Manufacturer

4 - Manufacturer and Device Class

5 - Manufacturer, Device Class, and Device Subclass

6 - Manufacturer, Device Class, Device Subclass, and Device Model

The format name syntax is similar to that used for SNVT types, except that the type name starts with "UNVT" instead of "SNVT". For example:

```
FORMAT:#800001128000000[4].UNVT_date_event
```

4.4.4 User Configuration Property Type (UCPT) Device Resource Files

This is a set of files that describes the data structures within UCPTs and also describes the formats used to display UCPT data. On the *i.*LON 100, these files may be found in the directory `/root/lonworks/types`, and are named BAS_Controller`.ENU`, BAS_Controller`.TYP`, BAS_Controller`.FMT` and BAS_Controller`.FPT`.

Echelon added these UCPTs for configuration properties used by *i.*LON 100 applications that have no SCPT definition. You can browse the UCPT resource files online at http://types.echelon.com.

4.5 Data Formatting

In order to keep the *i*.LON SOAP/XML interface neutral across regions, most of the rules for formatting data, which would normally be changeable in LNS, are not changeable on the *i*.LON 100. The one exception is the support of measurement system locale which is new in version 1.1. The following list describes the various regional settings used by the *i*.LON 100 SOAP / XML interface:

Decimal Symbol – Always period "."

Precision – Single floats always use 7 digits of precision, including digits before and after the decimal point. Double floats always use 14 digits of precision. For the rest of the base types, precision is determined by the type definition

Digit Grouping Symbol – Always comma ","

Digit Grouping – Always in the form "123,456,789"

Negative Sign Symbol – Always the minus sign "-"

Negative Number Format – Always "-1.1"; negative symbol in front, and no space between the symbol and the number

List Separators – If the format uses the localized list separator symbol verticle bar "|", the *i*.LON 100 will replace it with comma ",". However, if you define a new type in the NodeBuilder Resource Editor which is a structure, array or union, the default list separator is space " ". The localized list separator must be explicitly specified in the format.

Measurement System – The *i*.LON 100 does not use localization settings for measurement system. The measurement system used to display a formatted value is determined by the UCPTformatDescription property of the data point. For example, if you have a data point whose format is defined as SNVT_temp_f#US, then the UCPTvalue written to the DataServerRead SOAP message will be in Farenheit. If that data point is an input to the AlarmGenerator, then the format of a property which specifies a comparison value, a delta or an offset like UCPThighLimit2Offset will also be in US units when you read it with the AlarmGeneratorGet function. Furthermore, you must use US units when setting the property with the AlarmGeneratorSet function. You should note that the value stored in the XML file will always be in SI units so that XML files may be shared between *i*.LON 100s. The rule used by the applications is that the format of the primary data point for the application instance determines the format of measurement system dependent properties, like offsets, comparison values and deltas.

4.6 List, Get, Set and Delete Functions

The SOAP interface for each *i.*LON 100 application contains a List function, a Get function, a Set function, and a Delete function. Together, these functions make up a symmetric interface. You can use the response from the List command as the input to the Get command. You can use the response from the Get command as the input to the Set command. This section provides an overview of this feature, and describes how you can take advantage of it when using the SOAP interface.

NOTE: The SOAP interface for some applications contains additional SOAP functions you can use to manage your XML files. These functions are listed later in this chapter, and described in detail in Chapters 5-13.

4.6.1 List Functions

Use the List function to retrieve a list of all items created for an application. For example, the AlarmGeneratorList function returns a list containing the index number, description, last update time and functional block name of each Alarm Generator that you have added to the *i.*LON 100, with custom SOAP applications or with the *i.*LON 100 Configuration Software. Similarly, the DataLoggerList function returns a list containing the index number, last update time, description and functional block name of each Data Logger that you have added to the *i.*LON 100.

4.6.2 Get Functions

Use the Get function to retrieve the configuration of any items or instances that you have added to an application. For example, you would use the AlarmGeneratorGet function to retrieve the configuration of an Alarm Generator. Or, you would use DataLoggerGet to retrieve the configuration of a Data Logger. You must reference the item whose configuration is to be retrieved by its index number, which is defined when the item is created.

Now, consider a scenario where you have used the AlarmGeneratorList function to retrieve a list containing the index number of each Alarm Generator that has been added to the *i.*LON 100. You could use the list as the input for the AlarmGeneratorGet function. The AlarmGeneratorGet function would return the configuration of all the items included in the list.

You can also use the Get function to retrieve the configuration of a single item, by supplying the index number assigned to the item when it was created as input.

4.6.3 Set Functions

You can use the Set function to write to each of the XML files described in the previous section. When you invoke the Set function for an application for the first time, the associated XML file will be created in the /root/config/software directory of the *i.*LON 100, if it has not already been created. All data defined in the input passed to the function will be added to the XML file. Following this, you can use the Set function to add more data to the XML file, or to overwrite existing data.

For example, the first time an application invokes the AlarmGeneratorSet function, the AlarmGenerator.XML file will be created in the /root/config/software directory of the *i.*LON 100 (if it has not already been created by another application). The file will contain an element for each Alarm Generator defined in the input passed to the function, as well as the global configuration properties defined in the input passed to the function.

After its initial invocation, you can use the AlarmGeneratorSet function to overwrite the values of the global properties defined for the Alarm Generator application. You can also use it to add new Alarm Generators to the XML file, or to overwrite the configuration of exisiting Alarm Generators.

Each time you create an Alarm Generator (or any item or instance of an *i.*LON 100 application) using the Set method, the item will be assigned an index number. You will use that index number to identify that Alarm Generator when writing to its configuration later, or when referencing it from other functions.

When using the Set function to create an item such as an Alarm Generator, you shoud consider using output supplied by the corresponding Get function as the basis for your input. The following procedure describes how you might do so using the Alarm Generator functions. You could use this algorithm when programming any of the *i.*LON 100 applications.

1) Invoke the AlarmGeneratorList function to generate a list of Alarm Generators that have been added to the *i.*LON 100. This list includes the index number of each Alarm Generator.

2) Invoke the AlarmGeneratorGet function, using the list returned by the AlarmGeneratorList function as the input. The function will return the configuration of each Alarm Generator included in the list output.

3) Review the output from step 2, and choose an Alarm Generator to serve as your "default" Alarm Generator. The AlarmGeneratorGet output for this Alarm Generator will serve as the basis for the next Alarm Generator you create. Modify the values of each property in the string returned by AlarmGeneratorGet to match the configuration you want for the new Alarm Generator. This will be more efficient than building the input string for the Set function from scratch.

   NOTE: You must increment the index number assigned to the Alarm Generator, or remove the index number property from the string created in this step, when using this algorithm. Otherwise, the next step of this procedure will overwrite the configuration of the default Alarm Generator you have chosen. Chapters 5-13 describe this in more detail.

4) Invoke the AlarmGeneratorSet function, using the modified string created in Step 3 as input. The new item is successfully created, without recreating an input string that defines an entire Alarm Generator configuration from scratch, and with minimal risk of format errors. Chapters 5-13 will clarify the benefits of this algorithm.

4.6.4 Delete Functions

Use the Delete functions to delete items from an application. For example, use the AlarmGeneratorDelete function to delete an Alarm Generator. Or, use the DataServerDelete function to delete a data point.

You must reference the item to be deleted by its index number in the input you supply to the function.

4.7 Performance Issues

The *i.*LON 100 contains 32 MB of RAM, which allows for complicated application configurations and extensive network use. However, even with this amount of memory, it is still possible for very high levels of network traffic to the *i.*LON 100, especially using the SOAP interface, to eventually exhaust its memory. This could result in delays in network access of the *i.*LON 100, performance problems for the *i.*LON 100 applications, or in the worst case even a reboot of the *i.*LON 100.

If your *i.*LON 100 exhibits some of these symptoms, you should consider reducing the level of network traffic to it. The following numbers are guidelines that apply to the use of the *i.*LON 100's SOAP interface. While they are not absolute limits or guarantees of performance, they may be helpful to follow when attempting to manage the *i.*LON 100's network traffic load or troubleshoot a performance problem.

As a result, you should follow these guidelines when programming SOAP applications:

- Limit the number of data points referenced in a single Get or Read message to no more than 100. For more information, see Chapter 5, *Data Server*.
- Limit the number of alarm log records read in a single message to no more than 100. For more information on reading alarm log records, see *AlarmNotifierRead* on page 8-20.
- Limit the number of data log records read in a single message to no more than 150. For more information on reading data log records, see *DataLoggerRead* on page 6-11.
- If the combined XML file sizes for a given application exceed 100 KB, don't try to read all the configuration data for that application in a single Get message. This could potentially happen with the Event Scheduler application if all of its functional blocks were used, or possibly with the Alarm Notifier application.
- Don't send a request message larger than 100 KB. Some possible examples of this might be defining more than 100 NVL points in the Data Server in a single message with DataServerSet, or writing to 40 Alarm Notifiers in a single message with AlarmNotifierSet.
- Limit the number of simultaneous SOAP clients to no more than the number of web tasks specified in the WebParams.dat file on the *i.*LON 100. The default for this number is five.

4.8 Getting Started

Chapters 5-13 of this document provide more detailed information on the various applications of the *i.*LON 100, and describe the SOAP functions you can use to configure them. You should review Chapter 5 before attempting to program any of the *i.*LON 100 applications. This chapter introduces and describes the *i.*LON 100 Data Server, which manages the data points you will use to control your network. It describes each type of data point, and lists the different ways you can create these data points and add them to the Data Server.

Once you have created your data points and built the *i.*LON 100 Data Server, you will be able to reference those data points when configuring the various applications of the *i.*LON 100. Chapters 5-13 describe the applications of the *i.*LON 100, and the SOAP functions you can use to configure each one.

5 Data Server

The *i.*LON 100 uses the concept of a data point to map logical names to *i.*LON 100 system variables, network variables defined on the *i.*LON 100 LonTalk interface, and explicitly addressed network variables. This paradigm can be extended to handle data from other types of control networks as drivers for these buses become available.

Data points provide the *i.*LON 100 applications and Web server with a generic, open way to handle any piece of information in any type of network, such as the current value of a network variable in an LNS-managed network, or an explicit message in a closed LONWORKS system. This document describes how to use two kinds of data points:

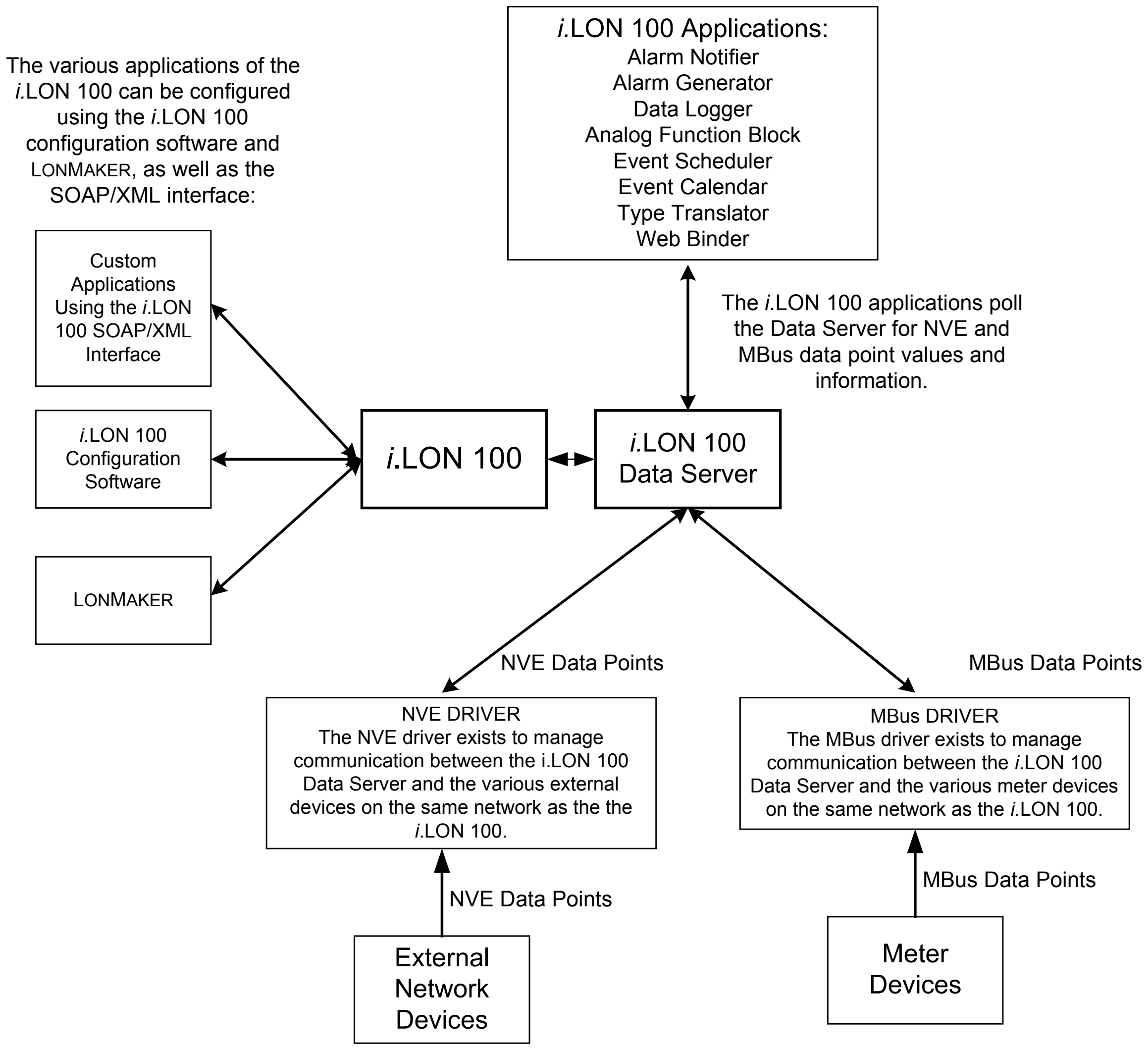
- NVL data points for network variables that are local to the *i.*LON 100
- NVC data points for *i.*LON 100 system variables that maintain constant values

The *i.*LON 100 Data Server handles all the details of these data point that are required by the various applications of the *i.*LON 100, such as how often a data point should be polled, its default value, its heartbeat, its current status, and its current value.

At the DataServer layer, all data points have the same set of properties, regardless of the network or device each data point is local to. This is made possible by the drivers that exist for each data point type, which handle communication between the Data Server and the network each data point is local to.

Use a standard network management tool for the particular data point type to configure each driver on the *i.*LON 100. For example, you could use an LNS-based network management tool to configure the NVL points on the *i.*LON 100. This layer of abstraction between the drivers and the DataServer provides a mechanism for all *i.*LON 100 applications to use data points of all types in the same way.

The Data Server also ensures that the configuration, status and value of each data point recognized by the tools you can use to configure the *i.*LON 100 remain synchronized with each other, and within the device each data point is local to. The tools you can use to configure the *i.*LON 100 include include custom SOAP applications, LONMAKER, and the *i.*LON 100 Configuration Software. The following diagram shows the relationship between the *i.*LON 100 Data Server and the different tools you can use to configure the *i.*LON 100 and its applications.



The various applications of the i.LON 100 can be configured using the i.LON 100 configuration software and LONMAKER, as well as the SOAP/XML interface:
i.LON 100 Applications:
Alarm Notifier
Alarm Generator
Data Logger
Analog Function Block
Event Scheduler
Event Calendar
Type Translator
Web Binder
Custom Applications Using the i.LON 100 SOAP/XML Interface
The i.LON 100 applications poll the Data Server for NVE and MBus data point values and information.
i.LON 100 Configuration Software
i.LON 100
i.LON 100 Data Server
LONMAKER
NVE Data Points
MBus Data Points
NVE DRIVER
The NVE driver exists to manage communication between the i.LON 100 Data Server and the various external devices on the same network as the the i.LON 100.
MBus DRIVER
The MBus driver exists to manage communication between the i.LON 100 Data Server and the various meter devices on the same network as the i.LON 100.
NVE Data Points
MBus Data Points
External Network Devices
Meter Devices

Two of the most important properties in the Data Server for any data point are the <UCPTpointStatus> and <UCPTvalue> properties. The <UCPTpointStatus> property represents the current status of the data point. The <UCPTvalue> property represents the current value of the data point. The Data Server updates these properties in real time, and they are very useful to many *i.*LON 100 applications.

For example, you could set up an Alarm Generator that will update the <UCPTpointStatus> of a data point to an alarm condition each time the <UCPTvalue> of that data point reaches a certain level. You could then set up an Alarm Notifier that will send out an alarm notification each time the <UCPTpointStatus> of the data point is updated to that condition. These applications are described in more detail later in this document.

A data point list is generated for each data point when it is created and added to the *i.*LON 100 Data Server. Once you have created the data points for your *i.*LON 100 and them to the Data Server, you can reference these data points when using *i.*LON 100 applications such as the Analog Function Block, Event Scheduler, Event Calendar, Type Translator, Alarm Generator, and Alarm Notifier. When any of these applications reference a data point, that application is added to the data point list for the data point, and the application will be notified each time the data point is updated. In this fashion, each application has current access to all the network information pertaining to the data points it is using.

This chapter describes how to create data points and add them to the Data Server.

NOTE: Echelon recommends that you restrict all networks to a maximum of 800 data points.

5.1 Data Server XML Files

The /root/config/software/dataserver directory of your *i.*LON 100 contains several XML files that will store the configuration of the data points in your Data Server. Table 7 describes these XML files.

Table 7 Data Server XML Files

File Name	Description
DP_NVL.XML	When a local NV for the *i.*LON 100 is created using LONMAKER, a corresponding NVL data point is automatically added to the Data Server. Its configuration can then be modified using the SOAP interface or the *i.*LON 100 Configuration Software, and it can be referenced from the *i.*LON 100 applications. The DP_NVL.XML file stores the configurations of the NVL data points in your Data Server. This file contains an entry for each static, dynamically created data point that has been added to the Data Server. **NOTE:** NVL data point must always be created using LONMAKER.
DP_NVC.XML	This file contains an entry for each NVC data point that has been added to the Data Server. An NVC data point represents a network variable that maintains a constant value. These data points can be created with the DataServerSet function, which is described later in this chapter, or with the *i.*LON 100 Configuration Software.

The following sections provide examples of each of these files. Guidelines and instructions to follow when modifying these files, manually or with the SOAP interface, follow the examples.

5.1.1 DP_NVL.XML

The DP_NVL.XML file is created automatically the first time the *i.*LON 100 boots. It will contain an <NVL> element for each static NV on the device. The properties contained within these elements define the configuration of an NVL data point, and are described later in this chapter. Each time you use LONMAKER to create a dynamic network variable for the *i.*LON 100, an <NVL> element for the associated data point will be added to this file.

You can modify a data point's configuration after it has been added to the *i.*LON 100 by manually editing this XML file, or by using the DataServerSet function. The sections following the example XML files provide guidelines and instructions to follow when doing so.

The following represents a sample DP_NVL.XML file for an *i.*LON 100 with four NVL data points.

```
<iLONDataServer>
  <DpNVL>
     <SCPTobjMajVer>1</SCPTobjMajVer>
     <SCPTobjMinVer>1</SCPTobjMinVer>
     <UCPTlastUpdate>2002-07-03T10:46:54Z</UCPTlastUpdate>
     <UCPTlifeTime>0</UCPTlifeTime>
  </DpNVL>
  <NVL>
     <UCPTindex>0</UCPTindex>
     <UCPTpointName>NVL_nvoAlarmFlag2</UCPTpointName>
     <UCPTlocation>iLON</UCPTlocation>
     <UCPTdescription />
     <UCPTformatDescription>SNVT_switch</UCPTformatDescription>
     <UCPTdpSize>2</UCPTdpSize>
     <UCPTbaseType>BT_STRUCT</UCPTbaseType>
     <UCPTunit>% of full level state code</UCPTunit>
     <SCPTmaxSendTime>0.0</SCPTmaxSendTime>
     <SCPTminSendTime>0.0</SCPTminSendTime>
     <SCPTmaxRcvTime>0.0</SCPTmaxRcvTime>
     <UCPTdefOutput>0.0 -1</UCPTdefOutput>
     <UCPTsettings>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0</UCPTsettings>
     <UCPTdirection>DIR_OUT</UCPTdirection>
  </NVL>
  <NVL>
     <UCPTindex>1</UCPTindex>
     <UCPTpointName>NVL_nviLevAlarm</UCPTpointName>
     <UCPTlocation>iLON</UCPTlocation>
     <UCPTdescription />
     <UCPTformatDescription>SNVT_alarm</UCPTformatDescription>
     <UCPTdpSize>29</UCPTdpSize>
     <UCPTbaseType>BT_STRUCT</UCPTbaseType>
     <UCPTunit></UCPTunit>
     <SCPTmaxSendTime>0.0</SCPTmaxSendTime>
     <SCPTminSendTime>0.0</SCPTminSendTime>
     <SCPTmaxRcvTime>0.0</SCPTmaxRcvTime>
     <UCPTdefOutput>0 0 0 0 0 0 0 AL_NO_CONDITION PR_LEVEL_0 0
        <0 0 0 0> 0/0/0/0:0:0:0 <0 0 0 0></UCPTdefOutput>
     <UCPTsettings>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0</UCPTsettings>
     <UCPTdirection>DIR_IN</UCPTdirection>
  </NVL>
```

```
<NVL>
   <UCPTindex>2</UCPTindex>
   <UCPTpointName>NVL_nvoDlClear</UCPTpointName>
   <UCPTlocation>iLON</UCPTlocation>
   <UCPTdescription />
   <UCPTformatDescription>SNVT_switch</UCPTformatDescription>
   <UCPTdpSize>2</UCPTdpSize>
   <UCPTbaseType>BT_STRUCT</UCPTbaseType>
   <UCPTunit>% of full level state code</UCPTunit>
   <SCPTmaxSendTime>0.0</SCPTmaxSendTime>
   <SCPTminSendTime>0.0</SCPTminSendTime>
   <SCPTmaxRcvTime>0.0</SCPTmaxRcvTime>
   <UCPTdefOutput>0.0 -1</UCPTdefOutput>
   <UCPTsettings>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0</UCPTsettings>
   <UCPTdirection>DIR_OUT</UCPTdirection>
</NVL>
<NVL>
   <UCPTindex>3</UCPTindex>
   <UCPTpointName>NVL_nviDeviceAlarm</UCPTpointName>
   <UCPTlocation>iLON</UCPTlocation>
   <UCPTdescription></UCPTdescription>
   <UCPTformatDescription>UNVT_alarm_2</UCPTformatDescription>
   <UCPTdpSize>31</UCPTdpSize>
   <UCPTbaseType>BT_STRUCT</UCPTbaseType>
   <UCPTunit />
   <SCPTmaxSendTime>0.0</SCPTmaxSendTime>
   <SCPTminSendTime>0.0</SCPTminSendTime>
   <SCPTmaxRcvTime>0.0</SCPTmaxRcvTime>
   <UCPTsettings>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0</UCPTsettings>
   <UCPTdirection>DIR_IN</UCPTdirection>
 </NVL>
</iLONDataServer>
```

5.1.2 DP_NVC.XML

The DP_NVC.XML file contains a list of <NVC> elements, one for each NVC data point that you have added to the Data Server. An NVC data point represents an *i.*LON 100 system variable that maintains a constant value.

Each element defines the configuration of an associated NVC data point. The properties that must be defined within each <NVC> element define the configuration of an NVC data point, and are described later in this chapter.

The following represents a sample DP_NVC.XML file for an *i.*LON 100 with two NVC data points. You can add NVC data points to the Data Server using the DataServerSet function, or by manually editing the XML file. The sections following the example XML files provide instructions and guidelines to follow when doing so.

```
<?xml version="1.0" ?>
  <iLONDataServer>
    <DpNVC>
       <SCPTobjMajVer>1</SCPTobjMajVer>
       <SCPTobjMinVer>1</SCPTobjMinVer>
       <UCPTlastUpdate>2002-05-07T15:25:10Z</UCPTlastUpdate>
       <UCPTlifeTime>0</UCPTlifeTime>
    </DpNVC>
    <NVC>
       <UCPTindex>0</UCPTindex>
       <UCPTpointName>NVC_nviConstant</UCPTpointName>
       <UCPTlocation />
       <UCPTdescription>Reference temperature</UCPTdescription>
       <UCPTformatDescription>SNVT_temp_p</UCPTformatDescription>
       <UCPTdpSize>2</UCPTdpSize>
       <UCPTunit>deg C</UCPTunit>
       <SCPTmaxSendTime>0.0</SCPTmaxSendTime>
       <SCPTminSendTime>0.0</SCPTminSendTime>
       <SCPTmaxRcvTime>20.0</SCPTmaxRcvTime>
       <UCPTdefOutput>0.00</UCPTdefOutput>
       <UCPTsettings>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0</UCPTsettings>
       <UCPTdirection>DIR_IN</UCPTdirection>
    </NVC>
    <NVC>
       <UCPTindex>1</UCPTindex>
       <UCPTpointName>NVC_nviTemp</UCPTpointName>
       <UCPTlocation />
       <UCPTdescription>SNVT_temp_f</UCPTdescription>
       <UCPTformatDescription>SNVT_temp_f</UCPTformatDescription>
       <UCPTdpSize>4</UCPTdpSize>
       <UCPTunit></UCPTunit>
       <SCPTmaxSendTime>0.0</SCPTmaxSendTime>
       <SCPTminSendTime>0.0</SCPTminSendTime>
       <SCPTmaxRcvTime>30.0</SCPTmaxRcvTime>
       <UCPTdefOutput>0</UCPTdefOutput>
       <UCPTsettings>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0</UCPTsettings>
       <UCPTdirection>DIR_IN</UCPTdirection>
    </NVC>
  </iLONDataServer>
```

5.2 Creating and Modifying the Data Server XML Files

The *i.*LON 100 generates all of the Data Server configuration files the first time it boots. The DP_NVL.XML file will contain an <NVL> element for each static NV on the device that has been created with LONMAKER. New NVL data points will be added to the DP_NVL.XML file automatically when more local NVs are created in LONMAKER. The DP_NVC.XML file will not contain any data point entries the first time the *i.*LON 100 boots.

You can use the DataServerSet function to add new NVC data points to the Data Server and to the DP_NVC.XMl file. You can also use DataServerSet to modify the configuration of existing NVL and NVC data points in the Data Server. The following section, *Data Server SOAP Interface*, describes how to use DataServerSet and the other SOAP functions provided for use with the Data Server.

You can manage the Data Server XML files manually using an XML text editor, and download them to the /root/config/software directory of the *i.*LON 100 via FTP. Echelon does not recommend this, as the *i.*LON 100 will require a reboot to read the configuration of the downloaded XML files. Additionally, the *i.*LON 100 performs error checking on all SOAP messages it receives before writing to the XML files. It will not perform error checking on any XML files you download via FTP, and so the application may not boot properly.

However, if you plan to create or modify any XML files manually, you should review the rest of this chapter first. This chapter describes the elements and properties in the Data Server configuration files that define each data point's configuration. For instructions on creating or modifying an XML file manually, see *Manually Modifying an XML Configuration File* on page 15-1.

5.2.1 Data Server SOAP Interface

The SOAP interface for the Data Server application includes seven functions. Table 8 lists and describes these functions. For more information, see the sections following Table 8.

Table 8 Data Server SOAP Functions

Function	Description
DataServerList	Use this function to list the index number, name and location of each data point that you have added to the Data Server. For more information, see *DataServerList* on page 5-10.
DataServerGet	Use this function to return the configuration of a data point. For more information, see *DataServerGet* on page 5-12.
DataServerSet	Use this function to create an NVC data point and add it to the Data Server, or to modify the configuration of an existing NVL or NVC data point. For more information, see *DataServerSet* on page 5-17.
DataServerRead	Use this function to read the current value of a data point, or group of data points, that has been added to the Data Server. For more information, see *DataServerRead* on page 5-19.
DataServerWrite	Use this function to write to the value of a data point, or group of data points. For more information, see *DataServerWrite* on page 5-24.

Function	**Description**
DataServerResetPriority	Use this function to reset the priority level assigned to a data point. For more information, see *DataServerResetPriority* on page 5-25.
DataServerDelete	Use this function to remove a data point from the Data Server. For more information, see *DataServerDelete* on page 5-28.

5.2.1.1 DataServerList

Use the DataServerList function to retrieve a list of data points that you have added to the *i.*LON 100 Data Server. The Data Server List function accepts an empty string as the contents of its <Data> parameter. If you supply an empty string as the <Data> parameter, the list returned by the function will contain an entry for every data point in the Data Server.

Alternatively, you can specify a subset of data points to be listed by filling the properties described in Table 9 into the <Data> parameter you supply to the function.

Table 9 DataServerList Input Properties

Parameter	Description
`<UCPTdataPointType>`	Enter the type of data point to be listed in the return string (NVL or NVC). Leave this property empty to display all data point types.
`<UCPTsetting>`	Optional. Enter a string of 16 comma-separated Boolean values. This string will be compared to the <UCPTsettings> string defined for each data point. If at least one set bit in this string matches the <UCPTsettings> string defined for a data point, then that data point will be included in the list returned by the function. The <UCPTsettings> property for a data point is defined when it is added to the Data Server, and can be written to with the DataServerSet function, which is described later in this chapter.
`<UCPTstartIndex>`	Enter the index number of the first data point to be listed in the return string.
`<UCPTcount>`	Enter the maximum number of data points to be included in the return string.

The example below requests that the function return a list of up to 50 NVL data points, starting with index number 0.

In this example, the function returns a <DpNVL> element at the beginning of the return string, because the <UCPTdataPointType> requested in the input is NVL. This contains global information about the data point type requested in the list. Or, if the property had been filled in as NVC, a <DpNVC> element would have been returned. Each of these elements includes the following child elements:

- <SCPTobjMajVer> and <SCPTobjMinVer>. Child elements identifying the major and minor version numbers of the firmware implementation the Data Server application is using.
- <UCPTlastUpdate>. A timestamp indicating the last time the Data Server was written to. This timestamp is expressed in UTC format, as per the ISO 8601 standard.
- <UCPTlifeTime>. This property defines how old (in seconds) the value of a data point of the specified type can be before the Data Server retrieves a new data value from the driver when an application requests the value of a given data point. If this parameter is set to 0, the values of the data points will be copied from the *i.*LON 100 Data Server when an application requests them, and no update will be requested from the driver. If this parameter is set to a positive value, the *i.*LON 100 Data Server will poll the driver for the current value of a data point each time an application requests it, and the time interval defined by the property has expired.

The interval resets each time the value of a data point is retrieved. By default this value is 0 for NVL data points, and 0 NVC data points. You can change this value by manually modifying it in the DP_NVL.XML or DP_NVC.XML configuration files. Note that you can also temporarily override this value each time you call the DataServerRead function. See the *DataServerRead* function later in this chapter for more information.

The <Result> parameter also includes an <NVL> element for each NVL data point that met the selection criteria defined in the input supplied to the function. Note that the function would return <NVC> element for each NVC data point that met the selection criteria defined in the function's input. The next section, *DataServerGet,* describes the properties included in each of these elements.

You could use the list of data point elements returned by this function as input for the DataServerGet function. The function would then return the configuration of each data point included in the list.

<Data> Parameter

```
<Data>
   <iLONDataServer>
      <UCPTdataPointType>NVL</UCPTdataPointType>
      <UCPTstartIndex>0</UCPTstartIndex>
      <UCPTcount>50</UCPTcount>
   </iLONDataServer>
</Data>
```

<Result> Parameter

```
<Result>
  <iLONDataServer>
    <DpNVL>
      <SCPTobjMajVer>1</SCPTobjMajVer>
      <SCPTobjMinVer>1</SCPTobjMinVer>
      <UCPTlastUpdate>2002-06-24T16:03:58Z</UCPTlastUpdate>
      <UCPTlifeTime>0</UCPTlifeTime>
    </DpNVL>
    <NVL>
      <UCPTindex>0</UCPTindex>
      <UCPTpointName>NVL_nviRequest</UCPTpointName>
      <UCPTlocation>iLON100</UCPTlocation>
    </NVL>
    <NVL>
      <UCPTindex>1</UCPTindex>
      <UCPTpointName>NVL_nvoStatus</UCPTpointName>
      <UCPTlocation>iLON100</UCPTlocation>
    </NVL>
    <NVL>
      <UCPTindex>2</UCPTindex>
      <UCPTpointName>NVL_nviTimeSet</UCPTpointName>
      <UCPTlocation>iLON100</UCPTlocation>
    </NVL>
    <NVL>
      <UCPTindex>3</UCPTindex>
      <UCPTpointName>NVL_nviDateEvent</UCPTpointName>
      <UCPTlocation>iLON100</UCPTlocation>
    </NVL>
  </iLONDataServer>
</Result>
```

5.2.1.2 DataServerGet

You can use the DataServerGet function to retrieve the configuration of any data point that you have added to the *i.*LON 100 Data Server. Each NVL data point to be returned must be identified by an <NVL> element within the <Data> parameter, and each NVC data point must be identified by an <NVC> element. You must reference the specific data point to be returned by its index number (UCPTindex) or its name (UCPTpointName) within each of these elements, as shown below.

You can request the configurations of any mixture of NVL or NVC data points in a single call to DataServerGet. The following example requests that the configuration of two data points be returned. One is referenced by its index number, and the other is referenced by its name. Note that the child element for each data point is called <NVL>: this would be <NVC> for an NVC data point.

NOTE: You should not attempt to retrieve the configuration of more than 100 data points with a single call to this function.

<Data> Parameter

```
<Data>
   <iLONDataServer>
      <NVL>
         <UCPTindex>0</UCPTindex>
      </NVL>
      <NVL>
         <UCPTpointName>NVL_nvoSwitch</UCPTpointName>
      </NVL>
   </iLONDataServer>
</Data>
```

<Result> Parameter

```
<iLONDataServer>
  <NVL>
      <UCPTindex>0</UCPTindex>
      <UCPTpointName>NVL_nviRequest</UCPTpointName>
      <UCPTlocation>iLON100</UCPTlocation>
      <UCPTdescription>Node request message input</UCPTdescription>
      <UCPTformatDescription>SNVT_obj_request</UCPTformatDescription>
      <UCPTdpSize>3</UCPTdpSize>
      <UCPTbaseType>BT_STRUCT</UCPTbaseType>
      <UCPTunit></UCPTunit>
      <SCPTmaxSendTime>0.0</SCPTmaxSendTime>
      <SCPTminSendTime>0.0</SCPTminSendTime>
      <SCPTmaxRcvTime>0.0</SCPTmaxRcvTime>
      <UCPTdefOutput>0,RQ_NORMAL</UCPTdefOutput>
      <UCPTsettings>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0</UCPTsettings>
      <UCPTdirection>DIR_IN</UCPTdirection>
  </NVL>
  <NVL>
      <UCPTindex>1</UCPTindex>
      <UCPTpointName>NVL_nvoSwitch</UCPTpointName>
      <UCPTlocation>MainBuilding\FirstFloor\Light</UCPTlocation>
      <UCPTdescription>Light switch</UCPTdescription>
      <UCPTformatDescription>SNVT_switch</UCPTformatDescription>
      <UCPTdpSize>6</UCPTdpSize>
      <UCPTbaseType>BT_STRUCT</UCPTbaseType>
      <UCPTunit></UCPTunit>
      <SCPTmaxSendTime>0.0</SCPTmaxSendTime>
      <SCPTminSendTime>0.0</SCPTminSendTime>
      <SCPTmaxRcvTime>0.0</SCPTmaxRcvTime>
      <UCPTdefOutput>100.0 1</UCPTdefOutput>
      <UCPTsettings>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0</UCPTsettings>
```

```
                    <UCPTdirection>DIR_OUT</UCPTdirection>
                   <UCPTvalueDef>
                      <OnValue>100.0 1</OnValue>
                      <OffValue>0.0 0</OffValue>
                    </UCPTvalueDef>
                </NVL>
              </iLONDataServer>
```

The DataServerGet function returns an element for each data point referenced in the <Data> parameter you supplied to the function. The properties included within each element are initially defined when the data point is added to the DataServer. You can write to them with the DataServerSet function. Table 10 describes these properties.

For more information on the DataServerSet function, see *DataServerSet* on page 5-17.

Table 10 DataServerGet Output properties

Property	Description
`<UCPTindex>`	The index number assigned to a data point must be in the range 0-32767. As mentioned earlier, you can use the DataServerSet function to create a new NVC data point, or to modify an existing NVL or NVC data point. If you do not specify an index number in the <Data> parameter you supply to DataServerSet, the function will create a new data point using the first available index number. If you specify an index number that is already being used, the function will overwrite the configuration of the data point using that index number with the settings defined in the <Data> parameter.
`<UCPTpointName>`	The name of a data point can be a maximum of 31 characters long, and must begin with the following prefixes: • NVL_ for an NVL data point • NVC_ for an NVC data point Once you have added a data point to the Data Server, you can not change its <UCPTpointName>. The <UCPTpointName> property for all data points must be unique, and must not contain any spaces. **Note:** The names assigned to NVL data points in the Data Server follow the naming convention NVL_[NAME], where [NAME] represents the progammatic name assigned to the NV when it was created with LONMAKER. You can determine the progammatic name of a network variable in LONMAKER by right-clicking it and selecting **Properties**.
`<UCPTlocation>`	An alphanumeric string of up to 128 characters that describes the location of the data point. This field is user-defined, and may be useful when organizing your data points by physical location or device.
`<UCPTdescription>`	A description of the data point. This can be a maximum of 228 characters long.

Property	Description
`<UCPTformatDescription>`	The format description of the data point. This determines many factors about the data point, including the type of values it takes, and its base type. This could be any standard (SNVT) format type included in the *i*.LON 100 resource files, or any user-defined (UNVT) format type included in resource files uploaded to the *i*.LON 100. For more information on the *i*.LON 100 resource files, see *i.LON 100 Resource Files* on page 4-8. The SNVT format types included in the *i*.LON 100 resource files are also listed and described in the SNVT Master List, which can be downloaded as a PDF by selecting the Documentation link on Echelon's Support Web site: http://www.echelon.com/support
`<UCPTunit>`	Unit text. This property is a string up to 228 characters long that describes the units the value of a data point is measured in. It should be filled in based on the network variable type assigned to the data point. A default value will be assigned to this property, if a unit for the network variable type chosen for the data point exists in the *i*.LON 100 resource files.
`<UCPTbaseType>`	This read-only property is assigned to the data point automatically, and is based on the point's <UCPTformatDescription>. It defines the base type of the data point, as defined in the base_type_t enumeration in the BAS_Controller resource files for the *i*.LON 100.
`<UCPTdpSize>`	Read-only. The size of the data point. This is determined based on the <UCPTformatDescription> selected for the data point.
`<SCPTmaxSendTime>`	This property applies to output data points. It defines the maximum amount of time that may elapse before the data point is updated on the network, if it is set to a non-zero value. For example, if a SNVT_temp value data point is changing by one degree every 10 seconds and this property is set to two seconds, the *i*.LON 100 will update the value of the data point on the network every two seconds, even though the value of the data point is not changing more than once every 10 seconds. The receiver can use this output as a heartbeat. The receiver will know something is wrong if he or she does not receive an update every two seconds.

Property	Description
`<SCPTminSendTime>`	This property applies to output data points, and defines the minimum amount of time that may elapse between data point updates if it set to a non-zero value. For example, if a SNVT_temp value data point is changing by one degree every half second and this value is set to two seconds, the data point will only be updated every two seconds with the latest value, even though the value changes more frequently than that. **NOTE:** If a data point update occurs before this time period elapses, that update will not be propagated over the network, unless the <SCPTmaxSendTime> property is defined. In that case, the update would be propagated over the network after the <SCPTmaxSendTime> period expires. If <SCPTmaxSendTime> is not set, the update would be lost. As a result, it is recommended that you set the <SCPTmaxSendTime> property.
`<SCPTmaxRcvTime>`	This property is used to control the maximum time that can elapse after an update to a bound network input, before another update occurs. If this period elapses without an update, the <UCPTpointStatus> of the data point will be updated to AL_OFFLINE. You could create an Alarm Notifier to trigger an alarm notification when this happens. For more information on Alarm Notifiers, see Chapter 8, *Alarm Notifier*. The valid range for this property is any value between 0.0 sec and 6,553.4 seconds. Setting <SCPTmaxRcvTime> to the default value of 0.0 disables the receive failure mechanism.
`<UCPTdefOutput>`	Optional. The value to be assigned to this data point after a power-up of the device or during an override of the functional block. For external data points and devices that operate as slaves, Echelon does not recommend that you define this property, as the value entered here will be sent to the external device after a power-on. **NOTE:** You can use DataServerSet to change this value in the Data Server. However, you must program your application to enforce the new value, as the *i.*LON 100 will continue to enforce the default value taken from the resource files.
`<UCPTminValue>`	Optional. This value is initially taken from the *i.*LON 100 resource files, if it exists for the data point type selected. This value represents the minimum value the data point can be updated to. **NOTE:** You can use DataServerSet to change this limit in the Data Server. However, you must program your application to enforce the new limit, as the *i.*LON 100 will continue to enforce the limit taken from the resource files.

Property	Description
`<UCPTmaxValue>`	Optional. This value is initially taken from the *i.*LON 100 resource files, if it exists for the data point type selected. This value represents the maximum value the data point can be updated to. **NOTE:** You can use DataServerSet to change this limit in the Data Server. However, you must program your application to enforce the new limit, as the *i.*LON 100 will continue to enforce the limit taken from the resource files.
`<UCPTinvalidValue>`	Optional. The invalid value for the data point. If the data point is updated and equals this value, the <UCPTpointStatus> of the data point will be updated to the AL_VALUE_INVALID. You could create an Alarm Notifier to trigger an alarm notification when this happens. For more information on Alarm Notifiers, see Chapter 8, *Alarm Notifier*. A default value for this property will be assigned based on the <UCPTformatDescription> selected. **NOTE:** You can use DataServerSet to change this value in the Data Server. However, you must program your application to enforce the new value, as the *i.*LON 100 will continue to enforce the value taken from the resource files.
`<UCPTsettings>`	A string of 16 comma-separated Boolean values. You could use these flags to determine access rights to the data point with your application. You can optionally specify a <UCPTsetting> string when you call the DataServerList function. This string will be compared to this property for each data point. If any bits in either string match, DataServerList will include the data point in its output. If no bits match, it will not include the data point. You could use this feature to restrict which users can view and access to certain data points.
`<UCPTvalueDef>`	This element can be used to create a series of value definitions for the data point. Each value definition is defined by a user-named attribute within the <UCPTvalueDef> element. These value definitions represent preset values that you can use to update the value of the data point when other *i.*LON 100 applications such as the Event Scheduler or the Alarm Notifier reference it. The sample <Data> parameter shown in this section defines two preset value definitions for the data point: OnValue and OffValue. **NOTE:** The values entered here must be in valid format, as defined by the network variable type assigned to the data point.

5.2.1.3 DataServerSet

Use the DataServerSet function to overwrite the configuration of an NVL or NVC data point, or to create an NVC data point and add it to the Data Server. Each NVL data point to be written to must be identified by an <NVL> element within the <Data> parameter, and each NVC data point must be identified by an <NVC> element. You must reference the specific data point to be written by its index number (UCPTindex) or its name (UCPTpointName) within each of these elements, as shown below.

The rest of properties that you must fill in within each data point element define the configuration of the data point within the Data Server. This set of properties is the same, whether you are creating a new data point or modifying an existing data point. The previous section, *DataServerGet*, describes each of these properties in detail.

It is important to realize that when you modify an existing data point with the DataServerSet function, any optional properties such as <UCTPminValue>, <UCPTmaxValue>, <UCPTdefOutput> and <UCPTinvalidValue> left out of the <Data> parameter will be erased. Old values will not be carried over, so you must fill in every property, or make sure that the data point is linked to a template defining the values any of these properties, when writing to an existing data point. Otherwise, these properties will be set to a null value.

The following example re-writes the configuration of a data point called NVL_nvo01Switch. Because this is an NVL data point, its new configuration is contained within an <NVL> element.

NOTE: You can create or write to multiple data points with a single call to DataServerSet. However, you should not attempt to create or write to more than 100 data points with a single call to this function. Additionally, to optimize the memory available to the *i.*LON 100, you should not have more than 800 data points in your network at any time.

<Data> Parameter	`<Data>`
	`    <iLONDataServer>`
	`        <NVL>`
	`            <UCPTindex>`**`200`**`</UCPTindex>`
	`            <UCPTpointName>`**`NVL_nvo01Switch`**`</UCPTpointName>`
	`            <UCPTlocation>`**`Light Kitchen`**`</UCPTlocation>`
	`            <UCPTdescription>`**`Lamp Kitchen 230V; 100W`**`</UCPTdescription>`
	`            <UCPTformatDescription>`**`SNVT_switch`**`</UCPTformatDescription>`
	`            <UCPTbaseType>`**`BT_STRUCT`**`</UCPTbaseType>`
	`            <UCPTunit>`**`State, %`**`</UCPTunit>`
	`            <SCPTmaxSendTime>`**`60`**`</SCPTmaxSendTime>`
	`            <SCPTminSendTime>`**`10`**`</SCPTminSendTime>`
	`            <SCPTdefOutput>`**`100.0 1`**`</SCPTdefOutput>`
	`            <UCPTminValue>`**`0.0 0`**`</UCPTminValue>`
	`            <UCPTmaxValue>`**`100.0 1`**`</UCPTmaxValue>`
	`            <UCPTinvalidValue>`**`0.0 -1`**`</UCPTinvalidValue>`
	`            <UCPTsettings>`**`1,1,1,1,0,0,0,0,0,0,0,0,0,1,1,1`**`</UCPTsettings>`
	`            <UCPTvalueDef>`
	`                <OnValue>`**`100.0 1`**`</OnValue>`
	`                <OffValue>`**`100.0 0`**`</OffValue>`
	`                <BypassValue>`**`50.0 1`**`</BypassValue>`
	`                <StdbyValue>`**`10.0 0`**`</StdbyValue>`
	`            </UCPTvalueDef>`
	`        </NVL>`
	`    </iLONDataServer>`
	`</Data>`

<Result> Parameter	`<Result>`
	`    <iLONDataServer>`
	`        <NVL>`
	`            <UCPTindex>`**`200`**`</UCPTindex>`
	`        </NVL>`
	`    </iLONDataServer>`
	`</Result>`

5.2.1.4 DataServerRead

You can use the DataServerRead function to read the value and status of any data point that you have added to the Data Server. There are two ways to reference the data points whose values and statuses are to be returned:

- You can reference each data point to be read by its index number or name in the <Data> parameter you supply to the function. If the specified input data point is a structure, you can specify the field whose value is to be returned below. For more information on this, and a sample <Data> string, see the *Requesting Data Points by Name and Index* section later in this chapter.
- You can reference a group of data points to be returned by the data point type, and by the last time the data points were updated. For more information on this, and a sample <Data> string, see the *Requesting Data Points by Type and Last Update Time* section later in this chapter.

The DataServerRead function will return a list of elements, one for each data point referenced by the <Data> parameter you supplied to the function. Each of these elements contains the current values of a group of properties and attributes associated with the associated data point. This includes the value and the priority level currently assigned to the data point. This is described in more detail in the in the *DataServerRead Output Properties* section later in this chapter.

5.2.1.4.1 Requesting Data Points by Name and Index

You can reference the data points to be returned by their index numbers or names in the <Data> parameter you supply to the function. This may be useful if you only need to request information for a small number of data points. You should not attempt to read more than 100 data points with a single call to this function.

The following example <Data> parameter requests that information for three separate data points be returned. Note that the name of each element within the <Data> parameter must match the data point type (NVL, NVC) of the data point referenced within it. Table 12, which follows the example, describes the properties returned by the function for each data point.

The information contained in the <Result> parameter for each data point returned by the function is described in the section later in this chapter.

NOTE: This example includes the optional <UCPTlifeTime> property. This defines how old the value of a data point can be, in seconds, before the Data Server retrieves a new data value from the driver when an application requests its value. If the property is set to 0, the values of the data points will be copied from the *i.*LON 100 Data Server when an application requests them, and no update will be requested from the applicable driver. If this parameter is set to a positive value, the *i.*LON 100 Data Server will poll the driver for the current value of a data point each time an application requests it, and the time interval defined by the property has expired. You can temporarily override the value of the <UCPTlifeTime> property stored in the *i.*LON 100 Data Server by passing it in to the DataServerRead message. In doing so, you can determine whether or not the values of the data points you are reading will be polled for this message. This may be useful if you are creating an application to monitor a device such as a thermometer, and do not necessarily need a current value. Set the property to 0, or any value greater than the current poll rate, if you do not want the values of the data points polled for this message.

<Data> Parameter	`<Data>`
	`     <iLONDataServer>`
	`     <UCPTlifeTime> </UCPTlifeTime>`
	`<NVC>`
	`              <UCPTindex>`**`0`**`</UCPTindex>`
	`         </NVL>`
	`         <NVC>`
	`             <UCPTpointName>`**`NVL_nvo03Switch`**`</UCPTpointName>`
	`         </NVL>`
	`         <NVL>`
	`             <UCPTindex>`**`9`**`</UCPTindex>`
	`             <UCPTfieldName>`**`state`**`</UCPTfieldName>`
	`         </NVL>`
	`     </iLONDataServer>`
	`</Data>`

5.2.1.4.2 *Requesting Data Points by Type and Last Update Time*

You can also reference the data points to be returned by their data point type, and time of last update. This may be useful if you want to see which data points were updated during a certain time period, or if you want to read the values of all data points of a certain type.

Table 11 DataServerRead Input Properties

Property	**Description**
`<UCPTdataPointType>`	Enter the type of data point you want to see information for. Enter NVL for NVL data points, or NVC for NVC data points. Leave this property empty to see all data point types in the <Result> parameter.
`<UCPTstart>` `<UCPTstop>`	Use these fields to specify a range for the last update time to the data points that will be returned by the function. Both parameters are optional. If you only specify a start time, the function will only return the data points whose value or status has been updated since the time specified. This is useful when an application only requires the latest updates of data points, and based on the activity of the data points requested, it can reduce the size of the response SOAP message. If you specify a start and stop time only data points whose last update time is between this interval will be returned by the function. If you only specify a stop time only data points whose last update time occurs before the stop time will be returned by the function. If you do not enter a start or stop time, the function will return all data points requested, up to the count specified. The <UCPTstart> and <UCPTstop> properties must be entered as timestamps in local time, with an appended time zone indicator that denotes the difference between local time and UTC. For more information on this format, see *Local Times and Coordinated Universal Time* on page 6-12.
`<UCPTcount>`	Use this field to specify the maximum number of data point entries the function will return. If this property is not filled in, the function will return all data points requested, or data points whose last update occurred within the interval defined by the <UCPTstart> and <UCPTstop> properties. **NOTE:** You should not attempt to read more than 100 data points with a single call to this function.
`<UCPTlifeTime>`	Optional. Use this property to determine whether the values of the data points requested in the <Data> parameter will be polled before they are returned. This is described in more detail earlier in the section.

The following example <Data> parameter requests that information for NVL data points updated in July of 2001 be returned. Because the <UCPTcount> property is set to 20, the function will include information for no more than 20 data points in the <Result> parameter.

The information contained in the <Result> parameter for each data point returned by the function is described in the next section of this chapter*, DataServerRead Output Properties*.

<Data> Parameter

```
<Data>
     <iLONDataServer>
             <UCPTlifeTime> </UCPTlifeTime>
             <UCPTdataPointType>NVL</UCPTdataPointType>
             <UCPTstart>2001-07-01T00:00:01.000+00:00</UCPTstart>
             <UCPTstop>2001-07-31T23:59:59.999+00:00</UCPTstop>
             <UCPTcount>20</UCPTcount>
     </iLONDataServer>
</Data>
```

5.2.1.4.3 DataServerRead Output Properties

The function returns an element for each data point referenced in the <Data> parameter you supplied to the function. Information for NVL data points will be included within an <NVL> element, and information for NVC data points will be included within an <NVC> data point.

<Result> Parameter

```
<Result>
     <iLONDataServer>
         <NVL>
             <UCPTindex>0</UCPTindex>
             <UCPTpointName>NVL_nvo01Switch</UCPTpointName>
             <UCPTpointUpdateTime>2001-07-24T01:47:22.000+01:00</UCPTpointUpdateTime>
             <UCPTvalue>0.0 0</UCPTvalue>
             <UCPTvalueDef>OffValue</UCPTvalueDef>
             <UCPTunit>% of full level,state code</UCPTunit>
             <UCPTpointStatus>AL_NO_CONDITION</UCPTpointStatus>
             <UCPTpriority>250</UCPTpriority>
         </NVL>
         <NVL>
             <UCPTindex>10</UCPTindex>
             <UCPTpointName>NVL_nvo03Switch</UCPTpointName>
             <UCPTpointUpdateTime>2001-07-24T01:47:22.000+01:00</UCPTpointUpdateTime>
             <UCPTvalue>33.0 0</UCPTvalue>
             <UCPTunit>% of full level,state code</UCPTunit>
             <UCPTpointStatus>AL_NO_CONDITION</UCPTpointStatus>
             <UCPTpriority>250</UCPTpriority>
         </NVL>
         <NVL>
             <UCPTindex>9</UCPTindex>
             <UCPTpointName>NVL_nvo02Switch</UCPTpointName>
             <UCPTfieldName>state</UCPTfieldName>
             <UCPTpointUpdateTime>2001-07-24T01:47:22.000+01:00</UCPTpointUpdateTime>
             <UCPTvalue>1</UCPTvalue>
             <UCPTunit>state code</UCPTunit>
             <UCPTpointStatus>AL_NO_CONDITION</UCPTpointStatus>
             <UCPTpriority>250</UCPTpriority>
         </NVL>
     </iLONDataServer>
</Result>
```

The following table describes the properties that are included in each of <NVL> or <NVC> element.

Table 12 DataServerRead Output Properties

Property	Description
`<UCPTindex>`	The index number assigned to the data point.
`<UCPTpointName>`	The name of the data point.

Property	Description
`<UCPTfieldName>`	If the value of a field was requested in the <Data> parameter, this property contains the name of the field.
`<UCPTpointUpdateTime>`	A timestamp indicating the last time the value of the data point was updated. This timestamp is expressed in local time, with an appended time zone indicator that indicates the difference between local time and Coordinated Universal Time (UTC). UTC is the current term for what was commonly referred to as Greenwich Meridian Time (GMT). Zero (0) hours UTC is midnight in Greenwich England, which lies on the zero longitudinal meridian. Universal time is based on a 24 hour clock, therefore, an afternoon hour such as 4 pm UTC would be expressed as 16:00 UTC. The timestamp uses the following format: [YYYY-MM-DD]T[HH:MM:SS.MSS]+/-[HH:MM] The first segment of the timestamp [YYYY-MM-DD] represents the date. The second segment (T[HH:MM:SS.MSS]) of the timestamp represents the local time, expressed in hours, minutes, seconds and milliseconds. The third segment (+/-[HH:MM]) represents the difference between the local time listed in the second segment and UTC. This segment begins with a + or a -. The + indicates that the local time is ahead of UTC, the - indicates the local time is behind UTC. Consider the following example: 2002-08-13T10:24:37.111+02:00 This timestamp indicates a local date and time of 10:24 AM and 37.111 seconds, on August 13, 2002. Because the third part of the segment reads +02:00, we know the local time here is 2 hours ahead of UTC.
`<UCPTvalue>`	The current value of the data point.
`<UCPTvalueDef>`	The value definition currently being used by the data point. Value definitions represent preset values. They can be created with the *i.*LON 100 Configuration Software, or the DataServerSet function. You can use these value definitions to update the value of the data point other *i.*LON 100 applications such as the Event Scheduler or the Alarm Notifier reference it.
`<UCPTunit>`	Unit type. This property is configured based on the network variable type of the data point.
`<UCPTpointStatus>`	The current status of the data point. This can be used when setting up Alarm Generators and Alarm Notifiers with the *i.*LON 100. For more information on these applications, see Chapter 7, *Alarm Generator*, and Chapter 8, *Alarm Notifier*.
`<UCPTpriority>`	The priority level currently assigned to the data point (0-255). The priority level of a data point determines which applications have write access to it. You can modify the value of this property with the DataServerWrite or DataServerResetPriority functions. For more information on priority levels, see *Data Point Values and Priority Levels* on page 3-5.

5.2.1.5 DataServerWrite

A data point's value and priority level are initially set when the data point is added to the Data Server. The value is set to the value established for the <UCPTdefOutput> property for the data point, and the priority defaults to the lowest priority level (255).

You can use the DataServerWrite function to update the value and priority of one or more data points at a time. Each NVL data point to be written to must be identified by an <NVL> element within the <Data> parameter, and each NVC data point must be identified by an <NVC> element. The rest of this section describes the information that must be filled in within each of these elements.

You must reference the specific data point to be written to by their index numbers (UCPTindex) or point names (UCPTpointName) within each of these elements.

If the data point is a structure, you can also specify the field whose value should be written to by the function by filling in the <UCPTfieldName> property. In this case, you may also want to fill in the <UCPTpropagate> property. If you assign the default value 1 to this property, the change you make to the data point will be propagated to the network. If you assign value 0 to this property, the change will be made in the *i.*LON 100 Data Server, but it will not be propagated over the LONWORKS network. This may be useful if you are writing to the different fields of a structure within a call to DataServerWrite, and do not want to update the structure over the network until all fields have been written by the function.

The priority level specified for each data point is set by the <UCPTpriority> property. This must be a higher priority than that used by the last application to write to the data point. If it is not, the data point will not be successfully updated. For more information on priority levels, see *Data Point Values and Priority Levels* on page 3-5.

You can write to the value of the data point using either a value definition (UCPTvalueDef), or an actual value (UCPTvalue). The example <Data> parameter below shows both of these options. You should not attempt to write to more than 100 data points with a single call to this function.

The following sample <Data> parameter writes to three data points. Note that the name of each element within the <Data> parameter must match the data point type (NVL or NVC) of the data point referenced within it.

<Data> Parameter

```
<Data>
     <iLONDataServer>
         <NVL>
             <UCPTpointName>NVL_nvo01Switch</UCPTpointName>
             <UCPTvalueDef>OffValue</UCPTvalueDef>
             <UCPTpriority>25</UCPTpriority>
         </NVL>
         <NVL>
             <UCPTindex>5</UCPTindex>
             <UCPTfieldName>state</UCPTfieldName>
             <UCPTvalue>1</UCPTvalue>
             <UCPTpriority>50</UCPTpriority>
             <UCPTpropagate>0</UCPTpropagate>
         </NVL>
     </iLONDataServer>
</Data>
```

<Result> Parameter	`<Result>`
	`        <iLONDataServer>`
	`            <NVL>`
	`                <UCPTindex>`**`8`**`</UCPTindex>`
	`            </NVL>`
	`            <NVL>`
	`                <UCPTindex>`**`10`**`</UCPTindex>`
	`                <UCPTfieldName>`**`state`**`</UCPTfieldName>`
	`            </NVL>`
	`        </iLONDataServer>`
	`    </Result>`

5.2.1.5.1 *DataServerWrite Fault Responses*

As of version 1.1, the *i.*LON 100 will return a detailed fault response message if it fails to update the value of any of the data points specified in the <Data> parameter. Consider a case where you specify input to update the values of 3 data points called NVL_myNetworkVar_1, NVL_myNetworkVar_2, and NVL_myNetworkVar_3. If the *i.*LON 100 is unable to update any one of the 3 data points for any reason, none of the data points will be updated, and a fault response message will be returned that indicates which data points the *i.*LON 100 was unable to update, and why.

The fault message will be returned using the same error reponse format as described in the *SOAP Error Responses* section in Chapter 2 of this document. However, the SOAP envelope will contain an additional element called the `<SOAP-ENV:detail>` after the `<SOAP-ENV:faultstring>` element. The `<SOAP-ENV:detail>` will contain a child-element for each data point that the *i.*LON 100 was unable to update.

In the following example, the *i.*LON 100 is unable to update the NVL_myNetworkVar_1 and NVL_myNetworkVar_2 data points. The `<SOAP-ENV:detail>` element includes a child element for each of these data points containing properties identifying the data point (UCPTindex, UCPTpointName, UCPTdataPointType) and the error code and description (faultcode and faulstring) that describes why the *i.*LON 100 was unable to update the data point. Remember that the value of the NVL_myNetworkVar_3 data point would not be updated by the funtion in this case, even though the new value passed to the function for the data point is valid.

<Data> Parameter	`<Data>`
	`   <iLONDataServer>`
	`   <NVL>`
	`      <UCPTpointName>`**`NVL_myNetworkVar_1`**`</UCPTpointName>`
	`      <UCPTvalue>`**`500.0 1`**`</UCPTvalue>`
	`   </NVL>`
	`   <NVL>`
	`      <UCPTpointName>`**`NVL_myNetworkVar_2`**`</UCPTpointName>`
	`      <UCPTvalueDef>`**`AUTO`**`</UCPTvalueDef>`
	`   </NVL>`
	`   <NVL>`
	`      <UCPTpointName>`**`NVL_myNetworkVar_3`**`</UCPTpointName>`
	`      <UCPTvalue>`**`100.0 1`**`</UCPTvalue>`
	`   </NVL>`
	`   </iLONDataServer>`
	`</Data>`

<SOAP-ENV:detail> of Fault Response Message

```
<SOAP-ENV:detail>
   <iLONDataServer/>
      <AoDP/>
      <DP/>
         <UCPTindex/>421</UCPTindex/>
         <UCPTdataPointType/>NVL</UCPTdataPointType/>
         <UCPTpointName/>NVL_myNetworkVar_1</UCPTpointName/>
         <faultcode/>11</faultcode/>
         <faultstring/>Value cannot be formatted</faultstring/>
      </DP/>
      <DP/>
         <UCPTindex/>422</UCPTindex/>
         <UCPTdataPointType/>NVL</UCPTdataPointType/>
         <UCPTpointName/>NVL_myNetworkVar_2</UCPTpointName/>
         <faultcode/>11</faultcode/>
         <faultstring/>Value cannot be formatted</faultstring/>
      </DP/>
      </AoDP/>
   </iLONDataServer/>
</SOAP-ENV:detail>
```

5.2.1.5.2 DataServerWrite and the Web Binder Application

You can use the Web Binder application to create connections that allow direct data exchange over a TCP/IP network between two *i*.LON 100s, or between an *i*.LON 100 and any Web server that can communicate via SOAP messaging such as Apache or IIS. You can configure the Web Binder using the built-in Web pages included with the *i*.LON 100, as described in Chapter 1 of the *i*.LON *100 Internet Server User's Guide: Using the i.LON 100 Web Pages to Configure Applications and to Monitor and Control Data Points* document.

Once the WebBinder has been configured, the *i*.LON 100 will send a DataServerWrite SOAP message for each update of a source data point to the destination server. Thus, to create an application on the Web server to receive WebBinder updates from the *i*.LON 100, you only need to implement the DataServerWrite method. You should note that an application which can receive the DataServerWrite message from the *i*.LON 100 differs from an application that would use all of the other methods described in this manual in that it must be a "server-side" application rather than a client application.

You can find an example if such a server-side WebBinder application on the i.LON 100 area of Echelon's website at:

http://www.echelon.com/ilon

5.2.1.6 DataServerResetPriority

You can use the DataServerResetPriority function to reset the priority of a data point to 255, the lowest priority.

Each NVL data point to be reset must be identified by an <NVL> element within the <Data> parameter, and each NVC data point must be identified by an <NVC> element. You must reference the specific data point to be reset by its index number (UCPTindex) or its name (UCPTpointName) in each of these elements, as shown below.

The new priority level for each data point is established by the value of the <UCPTpriority> property. Priority levels can be numbered from 0 to 255, with 0 representing the highest priority and 255 representing the lowest priority. Once the priority level assigned to a data point has been reset to 255, all applications will be able to write to the value of that data point.

The priority level specified in the <Data> parameter must be a higher priority than the current priority assigned to the data point for it to be reset. For more information on priority levels, see *Data Point Values and Priority Levels* on page 3-5.

NOTE: You should not attempt to reset more than 100 data points with a single call to this function.

<Data> Parameter

```
<Data>
  <iLONDataServer>
    <NVL>
        <UCPTindex>9</UCPTindex>
        <UCPTpriority>215</UCPTpriority>
    </NVL>
    <NVL>
        <UCPTpointName>NVL_nviRequest</UCPTpointName>
        <UCPTpriority>220</UCPTpriority>
    </NVL>
  </iLONDataServer>
</Data>
```

<Result> Parameter

```
<Result>
  <iLONDataServer>
    <NVL>
        <UCPTindex>9</UCPTindex>
    </NVL>
    <NVL>
        <UCPTindex>0</UCPTindex>
    </NVL>
  </iLONDataServer>
</Result>
```

5.2.1.7 DataServerDelete

You can use the DataServerDelete function to remove a data point from the Data Server. Each NVL data point to be deleted must be identified by an <NVL> element within the <Data> parameter, and each NVC data point must be identified by an <NVC> element. You must reference the specific data point to be deleted by its index number (UCPTindex) or its name (UCPTname) in each of these elements, as shown below.

The deletion of NVL data points requires two steps. NVL data points must be deleted from LONMAKER before they are deleted with this function. When you delete a local NV using LONMAKER, the status of the associated NVL data point in the Data Server will be set to AL_CONSTANT. In version 1.0 of the SOAP/XML interface, the index number of the associated NVL data point was also increased by 5000 to prevent errors from occurring when other applications attempt to reference the deleted data point before it is removed from the Data Server. In version 1.1, the incrementation of the index is no longer necessary.

NOTE: You should not attempt to delete more than 100 data points with a single call to this function.

<Data> Parameter	`<Data>`
	  `<iLONDataServer>`
	    `<NVL>`
	        `<UCPTindex>`**0**`</UCPTindex>`
	    `</NVL>`
	  `</iLONDataServer>`
	`</Data>`

<Result> Parameter	`<Result>`
	  `<iLONDataServer>`
	    `<NVL>`
	        `<UCPTindex>`**0**`</UCPTindex>`
	    `</NVL>`
	  `</iLONDataServer>`
	`</Result>`

6 Data Loggers

You can use Data Loggers to monitor activity on your network. Each Data Logger will record updates to a group of user-specified data points into a log file. The information recorded for each update includes the value and status that the data point was updated to.

Each *i.*LON 100 supports up to ten Data Loggers. 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. The Data Logger can save either type of log file in an ASCII-text (.csv file extension) or binary (.dat file extension) format.

You can specify the minimum amount of time that must elapse, and the minimum change in value required, between log entries for each data point your Data Logger is monitoring. When an update to a data point is logged, a subsequent update for that data point will not be logged until the minimum time period specified for the data point has elapsed, and the minimum value change specified for the data point has been met. If an input data points is updated more than once before the minimum time period has elapsed after a log entry has been recorded, the older values will be discarded. Only the most recent update will be recorded by the Data Logger when the minimum time period elapses. This allows you to throttle the data entry into a log.

You can also define a threhold 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 will enunciate that it is time to upload the log, and clear out some of the data. The Data Logger makes this enunciation by updating the Data Logger's alarm data point (called NVL_nvoDlAlarm[X], where X represents the index number assigned to the Data Logger) to the status AL_ALM_CONDITION. This feature may be useful when working with historical Data Loggers, which are disabled when they become full. You could create an Alarm Notifier to trigger an alarm notification when a log becomes full. For more information on Alarm Notifiers, see Chapter 9 of this document.

You can access the data in a log file by manually opening the log file, or by using the DataLoggerRead SOAP function. You can clear data from a log using the DataLoggerClear function, or by sending an update to the data point NVL_nviDlClear[X], where X represents the index number of the Data Logger to be affected. This is described in more detail later in the chapter.

6.1 DataLogger.XML

The DataLogger.XML file stores the configurations of each Data Logger that you have added to the *i.*LON 100. Each Data Logger is signified by a <Log> element in the XML file. The configuration properties contained in each <Log> element define the configuration of a Data Logger, and are described later in this chapter.

You can create newData Loggers using the DataLoggerSet SOAP function, or by manually editing the DataLogger.XML file. The sections following this example provide instructions and guidelines to follow when doing so.

The following represents a sample DataLogger.XML file for an *i.*LON 100 with three defined Data Loggers.

```
<?xml version="1.0" ?>
  <iLONDataLogger>
    <SCPTobjMajVer>1</SCPTobjMajVer>
    <SCPTobjMinVer>1</SCPTobjMinVer>
    <Log>
       <UCPTindex>0</UCPTindex>
       <UCPTlastUpdate>2002-02-12T14:36:51Z</UCPTlastUpdate>
       <UCPTdescription>Data Logger 0</UCPTdescription>
       <UCPTlogType>LT_CIRCULAR</UCPTlogType>
       <UCPTlogSize>100</UCPTlogSize>
       <UCPTlogFormat>LF_BINARY</UCPTlogFormat>
       <UCPTlogLevelAlarm>0.0</UCPTlogLevelAlarm>
       <Point>
          <UCPTindex>0</UCPTindex>
          <UCPTpointName>NVL_nviWHTot1</UCPTpointName>
          <UCPTlogMinDeltaTime>0.0</UCPTlogMinDeltaTime>
          <UCPTlogMinDeltaValue>0</UCPTlogMinDeltaValue>
          <UCPTpollRate>0</UCPTpollRate>
       </Point>
       <Point>
          <UCPTindex>1</UCPTindex>
          <UCPTpointName>NVL_nviWHTot2</UCPTpointName>
          <UCPTlogMinDeltaTime>0.0</UCPTlogMinDeltaTime>
          <UCPTlogMinDeltaValue>0</UCPTlogMinDeltaValue>
          <UCPTpollRate>0</UCPTpollRate>
       </Point>
     </Log>
     <Log>
       <UCPTindex>1</UCPTindex>
       <UCPTlastUpdate>2002-02-12T14:41:15Z</UCPTlastUpdate>
       <UCPTdescription>Data Logger 1</UCPTdescription>
       <UCPTlogType>LT_HISTORICAL</UCPTlogType>
       <UCPTlogSize>10</UCPTlogSize>
       <UCPTlogFormat>LF_TEXT</UCPTlogFormat>
       <UCPTlogLevelAlarm>30.0</UCPTlogLevelAlarm>
       <Point>
          <UCPTindex>0</UCPTindex>
          <UCPTpointName>NVL_nviDLTemp_f</UCPTpointName>
          <UCPTlogMinDeltaTime>0.0</UCPTlogMinDeltaTime>
          <UCPTlogMinDeltaValue>0</UCPTlogMinDeltaValue>
          <UCPTpollRate>0</UCPTpollRate>
        </Point>
     </Log>
     <Log>
        <UCPTindex>2</UCPTindex>
        <UCPTlastUpdate>2029-06-18T07:10:12Z</UCPTlastUpdate>
        <UCPTdescription>Data Logger 2</UCPTdescription>
        <UCPTlogType>LT_HISTORICAL</UCPTlogType>
        <UCPTlogSize>100</UCPTlogSize>
        <UCPTlogFormat>LF_TEXT</UCPTlogFormat>
        <UCPTlogLevelAlarm>30.0</UCPTlogLevelAlarm>
        <Point>
          <UCPTindex>0</UCPTindex>
          <UCPTpointName>NVL_AI_Analog</UCPTpointName>
          <UCPTlogMinDeltaTime>10.0</UCPTlogMinDeltaTime>
          <UCPTlogMinDeltaValue>0</UCPTlogMinDeltaValue>
          <UCPTpollRate>10</UCPTpollRate>
```

```
      </Point>
      <Point>
        <UCPTindex>1</UCPTindex>
        <UCPTpointName>NVL_nviWeekend</UCPTpointName>
        <UCPTlogMinDeltaTime>10.0</UCPTlogMinDeltaTime>
        <UCPTlogMinDeltaValue>0</UCPTlogMinDeltaValue>
        <UCPTpollRate>10</UCPTpollRate>
      </Point>
      <Point>
        <UCPTindex>2</UCPTindex>
        <UCPTpointName>NVL_nvoWeekday</UCPTpointName>
        <UCPTlogMinDeltaTime>10.0</UCPTlogMinDeltaTime>
        <UCPTlogMinDeltaValue>0</UCPTlogMinDeltaValue>
        <UCPTpollRate>10</UCPTpollRate>
      </Point>
  </Log>
</iLONDataLogger>
```

6.2 Creating and Modifying the DataLogger.XML File

You can create and modify the DataLogger.XML file with the DataLoggerSet SOAP function. The following section, *DataLogger SOAP Interface*, describes how to use DataLoggerSet and the other SOAP functions provided for the Data Logger application.

Alternatively, you can create and modify the DataLogger.XML file manually and download it to the *i.*LON 100 via FTP. Echelon does not recommend this, as the *i.*LON 100 will require a reboot to read the configuration of the downloaded file. Additionally, the *i.*LON 100 performs error checking on all SOAP messages it receives before writing to the XML file. It will not perform error checking on any XML files you download via FTP, and thus the application may not boot properly.

However, if you plan to create and manage the DataLogger.XML file manually, you should review the rest of this chapter first, as it describes the elements and properties in the XML file that define each Data Logger's configuration. For instructions on creating or modifying an XML file manually, see *Manually Modifying an XML Configuration File* on page 15-1.

6.2.1 DataLogger SOAP Interface

The SOAP interface for the Data Logger application includes six functions. Table 13 lists and describes these functions. For more information on each function, see the sections following Table 13.

Table 13 DataLogger SOAP Functions

Function	Description
DataloggerList	Use this function to generate a list of the Data Loggers that you have added to the *i.*LON 100. For more information, see *DataLoggerList* on page 6-5.
DataloggerGet	Use this function to retrieve the configuration of any Data Logger that you have added to the *i.*LON 100. For more information, see *DataLoggerGet* on page 6-6.
DataloggerSet	Use this function to create a new Data Logger, or to overwrite the configuration of an existing Data Logger. For more information, see *DataLoggerSet* on page 6-10
DataloggerRead	Use this function to read some, or all, of the log entries a Data Logger has recorded. For more information, see *DataLoggerRead* on page 6-11.
DataloggerClear	Use this function to remove some, or all, of the log entries a Data Logger has recorded from its log file. For more information, see *DataLoggerClear* on page 6-15.
DataloggerDelete	Use this function to delete a Data Logger. For more information, see *DataLoggerDelete* on page 6-16.

6.2.1.1 DataLoggerList

Use the DataLoggerList function to retrieve a list of the Data Loggers that you have added to the *i.*LON 100. The DataLoggerList function takes an empty string as its <Data> parameter, as in the example below.

The function returns the major and minor version numbers of the firmware implementation that the Data Logger application is using in the <Result> parameter. The <Result> parameter also includes a <Log> element for each Data Logger that you have added to the *i.*LON 100. The next section, *DataLoggerGet,* describes the properties included in each of these elements.

You could use the list of <Log> elements returned by this function as input for the DataLoggerGet function. The DataLoggerGet function would then return the configuration of each Data Logger included in the list.

<Data> Parameter

```
Empty String
```

<Result> Parameter

```
<Result>
   <iLONDataLogger>
      <SCPTobjMajVer>1</SCPTobjMajVer>
      <SCPTobjMinVer>1</SCPTobjMinVer>
      <Log>
         <UCPTindex>0</UCPTindex>
         <UCPTlastUpdate>2002-12-21T12:31:00Z</UCPTlastUpdate>
         <UCPTdescription>Light first Floor</UCPTdescription>
         <UCPTfbName>Data Logger- 0</UCPTfbName>
      </Log>
      <Log>
         <UCPTindex>1</UCPTindex>
         <UCPTlastUpdate>2002-12-21T12:31:01Z</UCPTlastUpdate>
         <UCPTdescription>Energy data</UCPTdescription>
         <UCPTfbName>Data Logger- 1</UCPTfbName>
      </Log>
      <Log>
         <UCPTindex>2</UCPTindex>
         <UCPTlastUpdate>2002-12-21T12:31:02Z</UCPTlastUpdate>
         <UCPTdescription>Light second Floor</UCPTdescription>
         <UCPTfbName>Data Logger- 2</UCPTfbName>
      </Log>
   </iLONDataLogger>
</Result>
```

6.2.1.2 DataLoggerGet

You can use the DataLoggerGet function to retrieve the configuration of any Data Logger that you have added to the *i.*LON 100. You must reference the Data Logger whose configuration is to be returned by its index number in the <Data> parameter you supply to the function, as in the example below.

<Data> Parameter	`<Data>`
	`  <iLONDataLogger>`
	`    <Log>`
	`      <UCPTindex>0</UCPTindex>`
	`    </Log>`
	`  </iLONDataLogger>`
	`</Data>`

<Result> Parameter	`<Result>`
	`  <iLONDataLogger>`
	`    <Log>`
	`      <UCPTindex>0</UCPTindex>`
	`      <UCPTlastUpdate>2002-02-12T14:36:51Z</UCPTlastUpdate>`
	`      <UCPTdescription>Temperature monitor</UCPTdescription>`
	`      <UCPTfbName>Data Logger- 0</UCPTfbName>`
	`      <UCPTlogType>LT_CIRCULAR</UCPTlogType>`
	`      <UCPTlogSize>100</UCPTlogSize>`
	`      <UCPTlogFormat>LF_BINARY</UCPTlogFormat>`
	`      <UCPTlogLevelAlarm>0.0</UCPTlogLevelAlarm>`
	`      <Point>`
	`        <UCPTindex>0</UCPTindex>`
	`        <UCPTpointName>NVL_nviWHTot1</UCPTpointName>`
	`        <UCPTlogMinDeltaTime>0.0</UCPTlogMinDeltaTime>`
	`        <UCPTlogMinDeltaValue>0</UCPTlogMinDeltaValue>`
	`        <UCPTpollRate>0</UCPTpollRate>`
	`      </Point>`
	`      <Point>`
	`        <UCPTindex>1</UCPTindex>`
	`        <UCPTpointName>NVL_nviWHTot2</UCPTpointName>`
	`        <UCPTlogMinDeltaTime>0.0</UCPTlogMinDeltaTime>`
	`        <UCPTlogMinDeltaValue>0</UCPTlogMinDeltaValue>`
	`        <UCPTpollRate>0</UCPTpollRate>`
	`      </Point>`
	`    </Log>`
	`  </iLONDataLogger>`
	`</Result>`

The function returns a <Log> element for each Data Logger referenced in the <Data> parameter you supplied to the function. The properties included in each element are initially defined when the Data Logger is created. You can write to them with the DataLoggerSet function. Table 14 describes these properties.

For more information on the DataLoggerSet function, see *DataLoggerSet* on page 6-10.

Table 14 DataLoggerGet Output Properties

Property	Description

Property	Description
`<UCPTindex>`	The index number assigned to the Data Logger must be in the range of 0-32,767. As mentioned earlier, you can use the DataLoggerSet function to create a new Data Logger, or to modify an existing Data Logger. If you do not specify an index number in the <Data> parameter you supply to DataLoggerSet, the function will create a new Data Logger using the first available index number. If you specify an index number that is already being used, the function will overwrite the configuration of the Data Logger using that index number with the settings defined in the <Data> parameter.
`<UCPTlastUpdate>`	A timestamp indicating the last time the configuration of the Data Logger was updated. This timestamp uses the following format: YYYY-MM-DD**T**HH:MM:SS**Z** The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Data Logger was last updated. The second segment (**T**HH:MM:SS) represents the time of day the configuration of the Data Logger was last updated, in UTC (Coordinated Universal Time). UTC is the current term for what was commonly referred to as Greenwich Meridian Time (GMT). Zero (0) hours UTC is midnight in Greenwich England, which lies on the zero longitudinal meridian. Universal time is based on a 24 hour clock, therefore, an afternoon hour such as 4 pm UTC would expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002.
`<UCPTfbName>`	The functional block name assigned to the Data Logger in LONMAKER. You can write to this property, but each time you use the *i*.LON 100 Configuration Software to view the Data Logger, it will be reset to match the functional block name defined in LONMAKER.
`<UCPTdescription>`	A user-defined description of the Data Logger. This can be a maximum of 228 characters long.
`<UCPTlogType>`	Either LT_HISTORICAL or LT_CIRCULAR. This indicates whether the log is a historical or circular. A historical data log stops recording data point updates when it is full. A circular data log removes older values when the log is full and it receives new updates.
`<UCPTlogSize>`	The amount of memory allocated to the log file, in kilobytes. Enter an integer value here. Please note that the total size of the log files for all Data Loggers (and Alarm Notifiers) on the *i*.LON 100 can not exceed the size of the flash memory stored in the *i*.LON 100. The *i*.LON 100 will stop writing to the log files when it only has 256 Kb of flash memory remaining.
`<UCPTlogFormat>`	Either LF_TEXT or LF_BINARY. This property indicates whether the log file the Data Logger uses will be an ASCII-text formatted CSV file (LF_TEXT), or use a proprietary binary format (LF_BINARY).

Property	Description
`<UCPTlogLevelAlarm>`	Enter a value between 0.0 and 100.0. The default value is 0.0. This value represents a percentage. When the volume of the Data Logger reaches this percentage, the status of the output data point for the Data Logger will be updated to the condition AL_ALM_CONDITION. The output data point for each Data Logger is called NVL_nvoDlLevAlarm[X], where X represents the index number assigned to the Data Logger. For example, if you enter 30.0 here, the data point would be updated when the log file has consumed 30% of the space allocated to it. You could create an Alarm Notifier to trigger an alarm notification each time one of your Data Loggers reaches this level. For more information on this, see Chapter 9, *Alarm Notifier*. You can determine the current log level of a Data Logger using the DataLoggerRead funtion, or by using the DataLoggerRead function to read the value field of the NVL_nviDlStatus[X] data point, where X represents the index number assigned to the Data Logger. The value assigned to the data point represents the percentage of the Data Logger's log file that has been used. You can clear out a log file using the DataLoggerClear function, or by updating the value assigned to NVL_nviDlClear[X], where X represents the index number assigned to the Data Logger. The value field you assign the data point when you update it reflects how much of the total log size will be cleared. For example, if your log is 50% full (out of 100kB), and you update the value of the data point to "30.0 1", then the application would go to the beginning of the log and clear out the first 30% of the log (in this case, 30K). You could use the DataServerWrite or DataPointWrite functions to update this data point's value.
`<Point>`	The data points the Data Logger will record updates for are defined by a list of <Point> elements. When any of the data points defined by these elements are updated, the Data Logger will record the updates into its log file. There are several properties you need to configure within each <Point> element that determine when an update to that data point will be logged. For descriptions of these properties, see Table 15 below. A Data Logger can record updates for as many data points as you want.

The data points a Data Logger monitors are defined by a list of <Point> elements. Table 15 describes the properties that should be defined within each <Point> element.

Table 15 DataLoggerGet <Point> Element Properties

Property	Description
`<UCPTpointName>`	The name of the data point to be monitored by the Data Logger, as defined in the *i*.LON 100 Data Server.
`<UCPTlogMinDeltaTime>`	The minimum amount of time, in seconds, that must pass between log entries for the data point. All updates will be logged if this value is 0.0, or not defined. This property has a maximum value of 214,748,364.0 seconds. The default is 0.0 seconds.

Property	Description
`<UCPTlogMinDeltaValue>`	This property applies to scalar data points only. Specify the change in value required for an entry to the log to be made. For example, if this property is set to 30.0, the value of the data point being monitored must change by at least 30.0 during an update for the change to be recorded by the Data Logger. All updates are logged if this value is 0.0, or not defined. This property has minimum and maximum floating point values of +/-3.402823466e+038. **NOTE:** If the format type used by the data point being monitored is SNVT_temp_p#US or SNVT_temp#US, then the value of this property returned by the DataLoggerGet function will be displayed using the SNVT_temp_f#US_diff format type. This rule applies to all formats that use the #US specifier.
`<UCPTpollRate>`	The poll rate for the Data Logger can be between 0 and 214,748,364.0 seconds. The Data Logger will check for updates to the data point at this interval. Echelon recommends that you set this to a value greater than or equal to the value specified for the <UCPTlogMinDeltaTime> property if you do not want to poll data before updates to the log are possible. If you use the default poll rate of 0 seconds, the Data Logger will record each updates to the data points it is monitoring into the log, assuming that the time period defined by the <UCPTlogMinDeltaTime> property has elapsed and the change in value specified by the <UCPTlogMinDeltaValue> property has been met. You should note that other *i.*LON 100 applications may cause the Data Server to poll this data point's value as well. The poll rate specified by these applications should be compatible with each other. For example, if an Alarm Generator is polling a data point every 15 seconds, and the Data Logger is polling that data point every 10 seconds, then the Data Server will have to poll the value of the data point every five seconds to ensure that each application gets a current value for each poll. It is important to note this as you set poll rates for various applications, as you may end up causing more polls than is efficient on your network. For example, if an Alarm Generator is polling a data point every 9 seconds and a Data Logger is polling a data point every 10 seconds, the Data Server would have to poll the data point every second to ensure that each application polls for a current value. This may create a significant amount of undesrired traffic.

6.2.1.3 DataLoggerSet

Use the DataLoggerSet function to create new Data Loggers, or to overwrite the configuration of existing Data Loggers. The Data Loggers to be created or written to are signified by a list of <Log> elements in the <Data> parameter. The properties you must define within each <Log> element are the same, whether you are creating a new Data Logger or modifying an existing Data Logger. The previous section, *DataLoggerGet*, describes these properties.

NOTE: When modifying an existing Data Logger, any optional properties left out of the input string will be erased. Old values will not be carried over, so you must fill in every property when writing to a Data Logger, even if you are not changing all of the values.

The first invocation of the DataLoggerSet function will generate the DataLogger.XML file in the /root/Config/Software directory of the *i.*LON 100, if the file does not already exist.

When creating or modifying a Data Logger with DataLoggerSet, you may want to use output from the DataLoggerGet function as the basis for your <Data> parameter. You would then only need to modify the values of each property to match the new configuration you want, as opposed to re-creating an entire string like the one shown below.

The following example creates a Data Logger that records all updates for two data points, one named NVL_nviWHTot1 and the other named NVL_nviWHTot2.

<Data> Parameter

```
<Data>
   <iLONDataLogger>
      <Log>
         <UCPTdescription>Data Logger 1</UCPTdescription>
         <UCPTfbName></UCPTfbName>
         <UCPTlogType>LT_CIRCULAR</UCPTlogType>
         <UCPTlogSize>100</UCPTlogSize>
         <UCPTlogFormat>LF_BINARY</UCPTlogFormat>
         <UCPTlogLevelAlarm>0.0</UCPTlogLevelAlarm>
         <Point>
            <UCPTindex>0</UCPTindex>
            <UCPTpointName>NVL_nviWHTot1</UCPTpointName>
            <UCPTlogMinDeltaTime>0.0</UCPTlogMinDeltaTime>
            <UCPTlogMinDeltaValue>0</UCPTlogMinDeltaValue>
            <UCPTpollRate>0.0</UCPTpollRate>
         </Point>
         <Point>
            <UCPTindex>1</UCPTindex>
            <UCPTpointName>NVL_nviWHTot2</UCPTpointName>
            <UCPTlogMinDeltaTime>0.0</UCPTlogMinDeltaTime>
            <UCPTlogMinDeltaValue>0</UCPTlogMinDeltaValue>
            <UCPTpollRate>0</UCPTpollRate>
         </Point>
      </Log>
   </iLONDataLogger>
</Data>
```

<Result> Parameter

```
<Result>
   <iLONDataLogger>
      <Log>
          <UCPTindex>3</UCPTindex>
      </Log>
   </iLONDataLogger>
</Result>
```

6.2.1.4 DataLoggerRead

Use the DataLoggerRead function to retrieve the entries in the log files generated by your Data Loggers. You can specify which log entries the function will return by filling the properties described in Table 16 into the <Data> parameter you supply to the function.

Table 16 DataLoggerRead Input Properties

Property	Description
`<UCPTindex>`	The index number of the Data Logger to return log entries for.
`<UCPTpointName>`	The name of the data point for which log entries are to be returned. If you do not fill in this property, the function will return log entries for all data points the Data Logger is monitoring.
`<UCPTcount>`	Use this field to specify the maximum number of log entries the function will return. If you do not fill in this property, the function will return all log entries for the applicable data point (or data points) that were logged during the interval defined by the <UCPTstart> and <UCPTstop> properties. **NOTE:** You should not attempt to read more than 150 log entries with a single call to this function.

`<UCPTstart>`
`<UCPTstop>`
	Use these fields to specify a time interval for the log entries to be returned. You can specify a start and stop time, or just a stop time.
	
	If you specify a start and stop time and the number of log entries during this interval exceeds the maximum defined by the <UCPTcount> property, the function will return the first group of log entries recorded during the interval.
	
	If you only specify a start time, the function will return entries from the log starting at the start time until it reaches the end of the log file, or until it has returned the maximum number of entries (as defined by the <Count> property).
	
	If you only specify a stop time and the number of log entries during this interval exceeds the maximum defined by the <UCPTcount> property, the function will return the group of entries from the stop time going backwards in the log until the maximum number of log entries have been returned. If the <UCPTcount> property was not defined, the function will return all log entries in the log, going backward from the stop time. This may be useful for applications that need to read the newest information logged.
	
	If you do not enter a start or stop time, the function will return all log entries for the applicable data points, up to the maximum.
	
	You must enter the <UCPTstart> and <UCPTstop> properties as timestamps in local time, with appended time zone indicators to denote the difference between local time and UTC. For more information on this format, see *Local Times and Coordinated Universal Time* on page 6-12.

6.2.1.4.1 Local Times and Coordinated Universal Time

The timestamps for the <UCPTstart> and <UCPTstop> properties conform to the ISO 8601 standard. They are expressed in local time, with appended time zone indicators that show the relationship to the Coordinated Universal Time (UTC).

UTC is an international time standard and is the current term for what was commonly referred to as Greenwich Meridian Time (GMT). Zero (0) hours UTC is midnight in Greenwich England, which lies on the zero longitudinal meridian. Universal time is based on a 24 hour clock, therefore, afternoon hours such as 4 pm UTC are expressed as 16:00 UTC. The timestamp uses the following format:

[YYYY-MM-DD]T[HH:MM:SS.MSS]+/-[HH:MM]

The first segment of the timestamp [YYYY-MM-DD] represents the date. The second segment (T[HH:MM:SS.MSS]) of the timestamp represents the local time, expressed in hours, minutes, seconds and milliseconds.

The third segment of the timestamp (+/-[HH:MM]) represents the difference between the local time listed in the second segment and UTC. This segment begins with a + or a -. The + indicates that the local time is ahead of UTC, and the - indicates the local time is behind UTC. Consider the following example:

2002-08-13T10:24:37.111+02:00

This timestamp indicates a local date and time of 10:24 AM and 37.111 seconds, on August 13, 2002. Because the third part of the segment reads +02:00, we know the local time here is 2 hours ahead of UTC.

6.2.1.4.2 Sample SOAP Message

The following example returns a list of up to three log entries made by the Data Logger with index number 2 between 10/27/2002 02:00 and 11/228/2002 14:30:00 for the NVL_nviDlCount2 data point.

<Data> Parameter

```
<Data>
  <iLONDataLogger>
    <Log>
      <UCPTindex>2</UCPTindex>
      <UCPTpointName>NVL_nviDlCount2</UCPTpointName>
      <UCPTstart>2002-01-27T02:00:00.000+01:00</UCPTstart>
       <UCPTstop>2002-11-28T04:30:00.000+01:00</UCPTstop>
      <UCPTcount>3</UCPTcount>
    </Log>
  </iLONDataLogger>
</Data>
```

<Result> Parameter

```
<Result>
  <iLONDataLogger>
    <Log>
      <UCPTindex>2</UCPTindex>
      <UCPTfileName>/root/data/log2.dat</UCPTfileName>
      <UCPTstart>2002-08-29T10:30:11.000-07:00</UCPTstart>
      <UCPTstop>2002-08-29T14:34:20.000-07:00</UCPTstop>
     <UCPTlastEvent>2003-08-29T14:34:20.000-07:00</UCPTlastEvent>
      <UCPTlogLevel>8.5</UCPTlogLevel>
     <UCPTtotalCount>57</UCPTtotalCount>
      <Element>
         <UCPTpointName>NVL_nviDlCount2</UCPTpointName>
         <UCPTlocation>iLON100</UCPTlocation>
         <UCPTlogSourceAddress>0.0</UCPTlogSourceAddress>
         <UCPTlogTime>2002-08-29T10:30:11.000-07:00</UCPTlogTime>
         <UCPTvalue>0</UCPTvalue>
         <UCPTunit>units (delta)</UCPTunit>
         <UCPTpointStatus>AL_NO_CONDITION</UCPTpointStatus>
      </Element>
      <Element>
         <UCPTpointName>NVL_nviDlCount2</UCPTpointName>
         <UCPTlocation>iLON100</UCPTlocation>
         <UCPTlogSourceAddress>1.3</UCPTlogSourceAddress>
         <UCPTlogTime>2002-08-29T10:31:00.000-07:00</UCPTlogTime>
         <UCPTvalue>5</UCPTvalue>
         <UCPTunit>units (delta)</UCPTunit>
         <UCPTpointStatus>AL_NO_CONDITION</UCPTpointStatus>
      </Element>
      <Element>
         <UCPTpointName>NVL_nviDlCount2</UCPTpointName>
         <UCPTlocation>iLON100</UCPTlocation>
         <UCPTlogSourceAddress>1.3</UCPTlogSourceAddress>
         <UCPTlogTime>2002-08-29T10:32:00.000-07:00</UCPTlogTime>
         <UCPTvalue>20</UCPTvalue>
         <UCPTunit>units (delta)</UCPTunit>
         <UCPTpointStatus>AL_NO_CONDITION</UCPTpointStatus>
      </Element>
    </Log>
  </iLONDataLogger>
</Result>
```

The DataLoggerRead function includes several global properties at the beginning of the <Result> parameter. These properties provide information about the Data Logger and the log file the entries were read from. Table 17 describes these properties.

Table 17 DataLoggerRead Global Output Properties

Property	Description
`<UCPTindex>`	The index number assigned the Data Logger.
`<UCPTfileName>`	The name of the log file the Data Logger is using.

`<UCPTstart>`
`<UCPTstop>`
	These properties represent timestamps indicating the log times of the first and last log entries in the log file. The timestamps are shown in local time, with appended time zone indicators showing the difference between local time and UTC. For more information on this, see *Local Times and Coordinated Universal Time* on page 6-12.

Property	Description
`<UCPTlastEvent>`	This property contains a timestamp indicating the last time an entry in the log file was deleted with the DataLoggerClear function, or the last time an entry in the log was modified with the DataLoggerWrite function. The timestamp is displayed in local time, with an appended time zone indicator that indicates the difference between local time and UTC. For more information on this, see *Local Times and Coordinated Universal Time* on page 6-12.
`<UCPTlogLevel>`	The volume of the log file that has been consumed, as a percentage. For example, the value 90.0 indicates that the log is 90% full.
`<UCPTtotalCount>`	This property contains the total number of entries contained in the data log read by the function.

The function also returns an <Element> element describing each log entry that met the selection criteria you defined in the <Data> parameter. Table 18 describes the properties listed within each of these elements.

Table 18 DataLogger Read <Element> Properties

Property	Description
`<UCPTpointName>`	The name of the data point updated.
`<UCPTlogSourceAddress>`	The source address of the data point. This is returned using the following format: [Subnet.Node]
`<UCPTlogTime>`	A timestamp indicating the time that the log entry was made. This timestamp is shown in local time, with an appended time zone indicator showing the difference between local time and UTC. For more information on this, see *Local Times and Coordinated Universal Time* on page 6-12.
`<UCPTvalueDef>`	Indicates the value definition currently being used by the data point. Value defintions are strings that represent preset values. They are created when a data point is added to the Data Server. For more information on this, see Chapter 5*, Data Server*. This property will be returned empty if the data point was not using a value definition after the update.
`<UCPTvalue>`	The value the data point was updated to.
`<UCPTunit>`	The unit type of the data point.
`<UCPTpointStatus>`	The status the data point was updated to.

6.2.1.5 DataLoggerClear

You can use the DataLoggerClear function to remove log entries from a Data Logger's log file. You can specify which Data Logger is to be affected, and which log entries will be removed, by configuring the properties described in Table 19 into the <Data> parameter you supply to the function.

NOTE: This function only deletes the log entries. You can delete the Data Logger itself with the DataLoggerDelete function.

Table 19 DataLoggerClear Input Properties

Parameter	Description
`<UCPTindex>`	The index number of the Data Logger to be affected.
`<UCPTpointName>`	The name of the data point whose log entries are to be deleted. If you do not fill in this property, the function will delete log entries for all data points that the Data Logger is monitoring.
`<UCPTcount>`	Use this property to specify the maximum number of log entries the function will delete. If you do not fill in this property, the function will delete all log entries for the applicable data point, or data points, that occurred within the interval defined by the <UCPTstart> and <UCPTstop> properties.
`<UCPTstart>` `<UCPTstop>`	Use these fields to specify a time interval for the log entries to be deleted. You can specify a start and stop time, or just a stop time. If you specify a start and stop time and the number of log entries during this interval exceeds the count entered, the function will delete the first group of log entries recorded during the interval. If you only specify a stop time and the number of log entries before this time exceeds the count entered, the function will delete the first group of log entries recorded before the stop time. If you do not enter a start or stop time, the function will delete all log entries for the applicable data points, up to the maximum. You must enter the <UCPTstart> and <UCPTstop> properties as timestamps in local time, with appended time zone indicators to denote the difference between local time and UTC. For more information on this format, see *Local Times and Coordinated Universal Time* on page 6-12.

The following call to DataLoggerClear deletes up to 200 log entries for data point NVL_nviDlCount2. These entries must have occurred between 1/27/2002 and 11/28/2002 (both at one hour ahead of UTC) to be deleted.

<Data> Parameter	
	```
	<Data>
	  <iLONDataLogger>
	    <Log>
	      <UCPTindex>2</UCPTindex>
	      <UCPTpointName>NVL_nviDlCount2</UCPTpointName>
	      <UCPTstart>2002-01-27T00:00:00.000+01:00</UCPTstart>
	      <UCPTstop>2002-11-28T00:00:00.000+01:00</UCPTstop>
	      <UCPTcount>3</UCPTcount>
	    </Log>
	  </iLONDataLogger>
	</Data>
	```

<Result> Parameter	
	```
	<Result>
	  <iLONDataLogger>
	    <Log>
	      <UCPTindex>2</UCPTindex>
	      <UCPTfileName>**/root/data/log2.dat**</UCPTfileName>
	      <UCPTstart>2002-08-29T10:32:00.000-07:00</UCPTstart>
	      <UCPTstop>2002-08-29T14:34:20.000-07:00</UCPTstop>
	      <UCPTlogLevel>8.5</UCPTlogLevel>
	    </Log>
	  </iLONDataLogger>
	</Result>
	```

The DataLoggerClear function includes several properties in the <Result> parameter. These properties provide information about the Data Logger and the log file affected by the function. Table 20 describes these properties.

Table 20 DataLoggerRead Output Properties

Property	Description
`<UCPTindex>`	The index number assigned to the Data Logger.
`<UCPTfileName>`	The name of the log file the Data Logger is using.
`<UCPTstart>` `<UCPTstop>`	These properties represent timestamps indicating the log times of the first and last log entries in the log file. The timestamps are shown in local time, with appended time zone indicators showing the difference between local time and UTC. For more information on this, see *Local Times and Coordinated Universal Time* on page 6-12.
`<UCPTlogLevel>`	The volume of the log file that has been consumed, as a percentage. For example, the value 90.0 indicates that the log is 90% full.

6.2.1.6 DataLoggerDelete

You can use the DataLoggerDelete function to delete a Data Logger. You must reference the Data Logger to be deleted by its index number in the <Data> parameter you supply to the function, as in the example below.

<Data> Parameter

```
<Data>
  <iLONDataLogger>
    <Log>
      <UCPTindex>0</UCPTindex>
    </Log>
  </iLONDataLogger>
</Data>
```

<Result> Parameter

```
<Result>
  <iLONDataLogger>
    <Log>
      <UCPTindex>0</UCPTindex>
    </Log>
  </iLONDataLogger>
</Result>
```

7 Alarm Generator

Use the Alarm Generator application to generate alarms based on the values of the data points in your network. Each time you create an Alarm Generator, you will select an input data point and a compare data point. The Alarm Generator will compare the values of these data points 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 (UCPTpointStatus) of the input data point will be updated to an alarm condition.

For example, you could select *GreaterThan* as the comparison function. The Alarm Generator would generate an alarm each time 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 application includes many other comparison functions like this, such as *Less Than*, *Less Than or Equal, Greater Than or Equal, Equal*, and *Null*. Each comparison function is described in detail later in the chapter.

The Alarm Generator application also includes a comparison function called *Limits*. 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. If the compare or input data points are updated, and the difference between their values exceeds any of the offset limits, an alarm will be generated.

You will define a hysteresis level for each alarm offset limit when you use the *Limits* 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. The relationship between the offset values, hysteresis levels, and alarm data points is described in more detail in the following sections.

All of the comparison functions have features like this that will allow you to throttle alarm generation. You can specifiy an interval (UCPTalarmSetTime) that must elapse between alarm generations for a data point. You can also define an interval (UCPTalarmClrTime) 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 UNVT_alarm2. The <UCPTpointStatus> of these data points, and of the input data point, will be updated to an alarm condition each time the Alarm Generator generates an alarm. The alarm data points are described in more detail later in the chapter.

You can use the Alarm Notifier application to generate e-mail messages when the alarm and input data points are updated to alarm conditions. For more information on this, see Chapter 9, *Alarm Notifier*.

7.1 AlarmGenerator.XML

The AlarmGenerator.XML file stores the configuration of the Alarm Generators that you have added to the *i.*LON 100. Each Alarm Generator is signified by an <Alarm> element in the XML file.

You can create new Alarm Generators using the AlarmGeneratorSet SOAP function, or by manually editing the AlarmGenerator.XML file, and rebooting the *i.*LON 100. The sections following this example provide instructions and guidelines to follow when doing so.

The following represents a sample AlarmGenerator.XML file for an *i.*LON 100 with one defined Alarm Generator.

```
<?xml version="1.0" ?>
  <iLONAlarmGenerator>
     <SCPTobjMajVer>1</SCPTobjMajVer>
     <SCPTobjMinVer>1</SCPTobjMinVer>
     <Alarm>
        <UCPTindex>0</UCPTindex>
        <UCPTlastUpdate>2002-06-20T12:37:53Z</UCPTlastUpdate>
        <UCPTdescription>Heating Control</UCPTdescription>
        <UCPTfbName>Alarm Generator- 0</UCPTfbName>
        <SCPTalrmIhbT>0 03:00:00.000</SCPTalrmIhbT>
        <UCPTalarmPriority>PR_LEVEL_1</UCPTalarmPriority>
        <UCPTpollOnResetDelay>0.0</UCPTpollOnResetDelay>
        <UCPTpollRate>0</UCPTpollRate>
        <UCPTalarm2Description>none</UCPTalarm2Description>
        <InputDataPoint>
           <UCPTpointName>NVL_DataValueA1</UCPTpointName>
        </InputDataPoint>
        <CompareDataPoint>
           <UCPTpointName>NVL_CompareValueA1</UCPTpointName>
        </CompareDataPoint>
        <AlarmDataPoint>
           <UCPTpointName>NVL_AlarmGenOut1</UCPTpointName>
        </AlarmDataPoint>
        <Alarm2DataPoint>
           <UCPTpointName>NVL_AlarmGenOut2</UCPTpointName>
        </Alarm2DataPoint>
        <UCPTcompFunction>FN_LIMIT</UCPTcompFunction>
        <UCPTalarmSetTime>0 00:30:00.000</UCPTalarmSetTime>
        <UCPTalarmClrTime>0 00:45:00.000</UCPTalarmClrTime>
        <UCPTlowLimit1Offset>5.0</UCPTlowLimit1Offset>
        <UCPTlowLimit2Offset>5.0</UCPTlowLimit2Offset>
        <UCPThighLimit1Offset>5.0</UCPThighLimit1Offset>
        <UCPThighLimit2Offset>5.0</UCPThighLimit2Offset>
        <SCPThystHigh1>50.00</SCPThystHigh1>
        <SCPThystHigh2>75.00</SCPThystHigh2>
        <SCPThystLow1>50.00</SCPThystLow1>
        <SCPThystLow2>75.00</SCPThystLow2>
      </Alarm>
  </iLONAlarmGenerator>
```

7.2 Creating and Modifying the AlarmGenerator.XML File

You can create and modify the AlarmGenerator.XML file with the AlarmGeneratorSet SOAP function. The following section, *Alarm Generator SOAP Interface*, describes how to use AlarmGeneratorSet, and the other SOAP functions provided for the Alarm Generator application.

Alternatively, you can create and modify the AlarmGenerator.XML file manually using an XML editor, and download the file to the *i.*LON 100 via FTP. Echelon does not recommend this, as the *i.*LON 100 will require a reboot to read the configuration of the downloaded file. Additionally, the *i.*LON 100 performs error checking on all SOAP messages it receives before writing to the XML file. It will not perform error checking on any XML files you download via FTP, and thus the application may not boot properly.

However, if you plan to create and manage the XML file manually, you should review the rest of this chapter first, as it describes the elements and properties in the XML file that define each Alarm Generator's configuration. For instructions on creating or modifying an XML file manually, see *Manually Modifying an XML Configuration File* on 15-1.

7.2.1 Alarm Generator SOAP Interface

The SOAP interface for the Alarm Generator application includes four functions. Table 21 lists and describes these functions. For more information, see the sections following Table 21.

Table 21 Alarm Generator SOAP Functions

Function	Description
AlarmGeneratorList	Use this function to generate a list of the Alarm Generators that you have added to the *i.*LON 100. For more information, see *AlarmGeneratorList* on page 7-4.
AlarmGeneratorGet	Use this function to retrieve the configuration of an Alarm Generator. For more information, see *AlarmGeneratorGet* on page 7-5.
AlarmGeneratorSet	Use this function to create a new Alarm Generator, or to overwrite the configuration of an exisiting Alarm Generator. For more information, see *AlarmGeneratorSet* on page 7-16.
AlarmGeneratorDelete	Use this function to delete an Alarm Generator. For more information, see *AlarmGeneratorDelete* on page 7-17.

7.2.1.1 AlarmGeneratorList

Use the AlarmGeneratorList function to retrieve a list of the Alarm Generators that you have added to the *i.*LON 100. The AlarmGeneratorList function takes an empty string as its <Data> parameter, as in the example below.

The function returns the major and minor version numbers of the firmware implementation that the Alarm Generator application is using in the <Result> parameter. The <Result> parameter also includes an <Alarm> element for each Alarm Generator that you have added to the *i.*LON 100. The next section, *AlarmGeneratorGet*, describes the properties included in each <Alarm> element.

You could use the list of <Alarm> elements returned by this function as input for the AlarmGeneratorGet function. The AlarmGeneratorGet function would then return the configuration of every Alarm Generator included in the list.

<Data> Parameter

```
Empty String
```

<Result> Parameter

```
<Result>
   <iLONAlarmGenerator>
       <SCPTobjMajVer>1</SCPTobjMajVer>
       <SCPTobjMinVer>1</SCPTobjMinVer>
       <Alarm>
              <UCPTindex>0</UCPTindex>
              <UCPTdescription>Light Control</UCPTdescription>
              <UCPTfbName>Alarm Generator- 0</UCPTfbName>
              <UCPTlastUpdate>2002-06-20T12:37:05Z</UCPTlastUpdate>
       </Alarm>
       <Alarm>
              <UCPTindex>1</UCPTindex>
              <UCPTdescription> Heating Control </UCPTdescription>
              <UCPTfbName> Alarm Generator- 1</UCPTfbName>
              <UCPTlastUpdate>2002-06-25T18:45:18Z </UCPTlastUpdate>
       </Alarm>
    </iLONAlarmGenerator>
</Result>
```

7.2.1.2 AlarmGeneratorGet

You can use the AlarmGeneratorGet function to retrieve the configuration of any Alarm Generator that you have added to the *i.*LON 100. You must reference the Alarm Generator whose configuration is to be retrieved by its index number in the <Data> parameter you supply to the function, as in the example below.

<Data> Parameter

```
<Data>
  <iLONAlarmGenerator>
   <Alarm>
       <UCPTindex>1</UCPTindex>
   </Alarm>
  </iLONAlarmGenerator>
</Data>
```

<Result> Parameter

```
<Result>
   <iLONAlarmGenerator>
       <Alarm>
              <UCPTindex>1</UCPTindex>
              <UCPTlastUpdate>2004-05-14T19:21:39Z</UCPTlastUpdate>
              <UCPTdescription>Heating Controller</UCPTdescription>
              <UCPTfbName>Alarm Generator- 1</UCPTfbName>
              <SCPTalrmIhbT>0 00:00:00.000</SCPTalrmIhbT>
              <UCPTalarmPriority>PR_LEVEL_1</UCPTalarmPriority>
              <UCPTpollOnResetDelay>0.0</UCPTpollOnResetDelay>
              <UCPTpollRate>0.0</UCPTpollRate>
              <UCPTalarm2Description>none</UCPTalarm2Description>
              <InputDataPoint>
                     <UCPTpointName>NVL_nviTemp0</UCPTpointName>
              </InputDataPoint>
              <CompareDataPoint>
                     <UCPTpointName>NVL_nviTemp1</UCPTpointName>
              </CompareDataPoint>
              <AlarmDataPoint>
                     <UCPTpointName>NVL_nvoAlarm0</UCPTpointName>
              </AlarmDataPoint>
              <Alarm2DataPoint>
                     <UCPTpointName>NVL_nvoAlarm2</UCPTpointName>
              </Alarm2DataPoint>
              <UCPTcompFunction>FN_LIMIT</UCPTcompFunction>
              <UCPTalarmSetTime>0 00:00:00.000</UCPTalarmSetTime>
              <UCPTalarmClrTime>0 00:00:00.000</UCPTalarmClrTime>
              <UCPTlowLimit1Offset>4.0</UCPTlowLimit1Offset>
              <UCPTlowLimit2Offset>8.0</UCPTlowLimit2Offset>
              <UCPThighLimit1Offset>4.0</UCPThighLimit1Offset>
              <UCPThighLimit2Offset>8.0</UCPThighLimit2Offset>
              <SCPThystHigh1>2.0</SCPThystHigh1>
              <SCPThystHigh2>2.0</SCPThystHigh2>
              <SCPThystLow1>2.0</SCPThystLow1>
              <SCPThystLow2>2.0</SCPThystLow2>
       </Alarm>
   </iLONAlarmGenerator>
</Result>
```

The function returns an <Alarm> element for each Alarm Generator referenced in the <Data> parameter you supplied to the function. The properties contained within each

<Alarm> element are initially defined when you create the Alarm Generator. You can write to them with the AlarmGeneratorSet function. Table 22 describes these properties.

When creating or writing to an Alarm Generator with the AlarmGeneratorSet function, all properties are mandatory unless otherwise noted. For more information on the AlarmGeneratorSet function, see *AlarmGeneratorSet* on page 7-16.

Table 22 AlarmGeneratorGet Output Properties

Property	Description
`<UCPTindex>`	The index number assigned to the Alarm Generator must be in the range of 0-32,767. As mentioned earlier, you can use the AlarmGeneratorSet function to create a new Alarm Generator, or to modify an existing Alarm Generator. If you do not specify an index number in the <Data> parameter you supply to AlarmGeneratorSet, the function will create a new Alarm Generator using the first available index number. If you specify an index number that is already being used, the function will overwrite the configuration of the Alarm Generator using that index number with the settings defined in the <Data> parameter.
`<UCPTlastUpdate>`	Optional. A timestamp indicating the last time the configuration of the Alarm Generator was updated. This timestamp uses the following format: YYYY-MM-DD**T**HH:MM:SS**Z** The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Alarm Generator was last updated. The second segment (**T**HH:MM:SS) represents the time of day the configuration of the Alarm Generator was last updated, in UTC (Coordinated Universal Time). UTC is the current term for what was commonly referred to as Greenwich Meridian Time (GMT). Zero (0) hours UTC is midnight in Greenwich England, which lies on the zero longitudinal meridian. Universal time is based on a 24 hour clock, therefore, an afternoon hour such as 4 pm UTC would expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002.
`<UCPTfbName>`	The functional block name assigned to the Alarm Generator in LONMAKER. You can write to this property, but each time you use the *i*.LON 100 Configuration Software to view the Alarm Generator, it will be reset to match the functional block name defined in LONMAKER.
`<UCPTdescription>`	Optional. A user-defined description of the Alarm Generator. This can be a maximum of 228 characters long.
`<SCPTalrmIhbT>`	The time period for which alarm generation is to be inhibited after the application is enabled. This period must be entered using the following format: DAYS HOURS:MINUTES:SECONDS.MILLISECONDS For example, 3 13:44:02.155 indicates a time period of 3 days, 13 hours, 44 minutes and 2.155 seconds.

Property	Description
`<UCPTalarmPriority>`	Specifies the alarm priority that will be reported in the *priority_level* field of the alarm data points for the Alarm Generator. The alarm priority is independent of the alarm type. For a list of valid alarm priorities, see *Alarm Priority Levels* on page 7-11.
`<UCPTpollOnResetDelay>`	The time period to wait after enabling or starting the application before polling the value of the input data point. This field has a range of 0.0-6553.0 seconds. When the default value of 0.0 seconds is used, the Alarm Generator will resume polling the input data point at the interval specified by the <UCPTpollRate> property immediately after a reset.
`<UCPTpollRate>`	The poll rate for the input and compare data points, in seconds. When this value is greater than 0, the Alarm Generator will poll the values of the input and compare data points each time this interval expires. This field has a range of 0-214,748,364 seconds. When this value is 0, the Alarm Generator will not poll the value of the input and compare data points, and will only check for alarm conditons when it receives event-driven updates to the data points. You should note that other *i.*LON 100 applications may cause the Data Server to poll this data point's value as well. The poll rate specified by these applications should be compatible with each other. For example, if an Alarm Generator is polling a data point every 15 seconds, and the Data Logger is polling that data point every 10 seconds, then the Data Server will have to poll the value of the data point every five seconds to ensure that each application gets a current value for each poll. It is important to note this as you set poll rates for various applications, as you may end up causing more polls than is efficient on your network. For example, if an Alarm Generator is polling a data point every 9 seconds and a Data Logger is polling a data point every 10 seconds, the Data Server would have to poll the data point every second to ensure that each application polls for a current value. This may create a significant amount of undesired traffic.
`<UCPTalarm2Description>`	Optional. Enter the description field of the UNVT_alarm2 data point selected for the Alarm Generator. This data point can be selected by defining the <Alarm2DataPoint> property. This description could include the value that increased and caused the alarm, an alarm or error code defined by the manufacturer, or the alarm limit. This can be a maximum of 22 characters long, and will be inserted in the description field of the UNVT_alarm2 data point each time an alarm is generated.

Property	Description
`<InputDataPoint>`	The input data point for this Alarm Generator. The data point must be referenced by its <UCPTpointName>. Each time this data point is updated, its value will be compared to the value of the compare data point using the comparison function defined by the <UCPTcompFunction> property. If the result of the comparison is True, an alarm will be generated. The <UCPTpointSatus> of this data point will be updated to the status AL_ALM_CONDITION when an alarm is generated, unless the <UCPTcompFunction> selected for the Alarm Generator is FN_LIMIT. In this case, the status will be updated to any of four alarm statuses, based on the offset limit that caused the alarm. For more information on this, see *Hysteresis Levels and Offset Limits* on page 7-13. You can register the input data point with the Alarm Notifier application to generate alarm notifications and e-mail messages each time it is updated to an alarm status. For more information on this, see *Alarm Notifier* on page 8-1.
`<CompareDataPoint>`	The compare data point for this Alarm Generator. The data point must be referenced by the <UCPTpointName> assigned to it in the Data Server, and must use the same format type as the input data point. The value of this data point will be compared to the value of the input data point each time either point is updated. Use an NVC data point as the compare data point if you want your alarm generator to generate alarms based on a constant value configured through software, as opposed to a live value taken from the network.

Property	Description

`<AlarmDataPoint>`
 	
`<Alarm2DataPoint>`
	Optional. These properties define the Alarm Generator's alarm data points. Each data point must be referenced by the <UCPTpointName> assigned to it in the Data Server. The data point chosen for the <AlarmDataPoint> must use the format type SNVT_alarm. The data point chosen for the <Alarm2DataPoint> must use the format type UNVT_alarm2.
	
	Use a SNVT_alarm data point if your system can handle this LonMark standard type for alarming. Use a UNVT_alarm2 data point if your system will require the additional information you can provide with the <UCPTalarm2Description> property. If your system can directly access the <UCPTpointStatus> property of the input data point, you may not need to use alarm data points, as your Alarm Generators will update the input data point to an alarm status each time they generate an alarm. You can read this property from a data point with the DataPointRead or DataServerRead functions.
	
	The <UCPTpointSatus> of the alarm data points will be updated to the status AL_ALM_CONDITION when an alarm is generated, unless the <UCPTcompFunction> is FN_LIMIT. In this case, the status will be updated to any of four alarm statuses, based on the offset limit that caused the alarm. For more information on this, see *Hysteresis Levels and Offset Limits* on page 7-13.
	
	You can register these alarm data points with the Alarm Notifier application to generate alarm notifications and e-mail messages each time they are updated to an alarm status. For more information on this, see *Alarm Notifier* on page 8-1.

`<UCPTcompFunction>`	Specifies the function that the Alarm Generator will use to compare the values of the input data point and the compare data point. For descriptions of the comparison functions you can use, see *Comparison Functions* on page 7-12.
`<UCPTalarmSetTime>`	Specifies the time period an alarm condition must exist before the Alarm Generator will consider it a valid alarm and generate an alarm. You must use the following format for this property: DAYS HOURS:MINUTES:SECONDS.MILLISECONDS For example, 0 1:12:12.333 indicates a time period of 1 hour, 12 minutes and 12.333 seconds. The maximum value you can enter is: 0 1:59:59.999 The default value is: 0 00:00:00.000

Property	Description
`<UCPTalarmClrTime>`	Specifies the time period to wait after the condition that caused an alarm has returned to normal status before the alarm will be cleared. You must use the following format for this property: DAYS HOURS:MINUTES:SECONDS.MILLISECONDS For example, 0 6:22:12.333 indicates a time period of 6 hours, 22 minutes and 12.333 seconds. The maximum value you can enter is: 0 1:59:59.999 The default value is: 0 00:00:00.000

`<UCPTlowLimit1Offset>`
 	
`<UCPTlowLimit2Offset>`
 	
`<UCPThighLimit1Offset>`
 	
`<UCPThighLimit2Offset>`
	Enter a scalar value for each of these properties. These values will be used as the offset limits for the Alarm Generator when the <UCPTcompFunction> property is set to FN_LIMIT. In this case, alarms will be generated when any of the following conditions are true:
	
	• Value of Input Data Point> Value of Compare Data Point +highLimit1Offset
	• Value of Input Data Point > Value of Compare Data Point +highLimit2Offset
	• Value of Input Data Point < Value of Compare Data Point – lowLimit1Offset
	• Value of Input Data Point < Value of Compare Data Point – lowLimit2Offset
	
	The value entered for <UCPThighLimit2Offset> must be greater than that entered for <UCPThighLimit1Offset>, and the value entered for <UCPTlowLimit2Offset> must be less than that entered for <UCPTlowLimit1Offset>. The default value for each property is 0. If any of these properties are left empty, they will not be used to check for alarm conditions. When you set these properties, you must also set the corresponding hysteresis properties, which are described later in the table.
	
	Each alarm condition caused by the offset properties will cause the <UCPTpointStatus> of the input data and alarm data points to be set to a different status. For more information on this, see see *Hysteresis Levels and Offset Limits* on page 7-13.
	
	NOTE: If you use the AlarmGeneratorGet function to return the configuration of an Alarm Generator whose input or compare data points use the format type SNVT_temp_p#US or SNVT_temp#US, then the values of these properties will be displayed using the SNVT_temp_f#US format. This rule applies to all formats that use the #US specifier.

Property	Description

`<SCPThystHigh1>`
`<SCPThystHigh2>`
`<SCPThystLow1>`
`<SCPThystLow2>`
	When an alarm occurs based on one of the offest limits described above, the value of the input data point must reach the hysteresis value for that limit before the alarm can be cleared, and another alarm can be generated based on that offset limit.
	
	This allows you to set up an Alarm Generator that will trigger an alarm once each time the value of the input data point reaches a certain level, as opposed to multiple times (which would occur each time the data point was updated and its value remained within the range specified by the offset limit).
	
	Enter a scalar value for each of these properties. These values define the hysteresis level that will be used for each alarm offset limit. For a more detailed description of the hysteresis fields and how they relate to the offset limit values, see *Hysteresis Levels and Offset Limits* on page 7-13.
	
	If you use the *i.*LON 100 Configuration Software to modify the configuration of an Alarm Generator after creating it with the SOAP/XML interface, all four hysteresis properties will be reset to the value chosen for <SCPThystHigh1>.
	
	NOTE: If you use the AlarmGeneratorGet function to return the configuration of an Alarm Generator whose input data point uses the format type SNVT_temp_p#US or SNVT_temp#US, then the values of these properties will be displayed using the SNVT_temp_f#US format. This rule applies to all formats that use the #US specifier.

7.2.1.2.1 Alarm Priority Levels

You can select a priority level for the Alarm Generator by filling in the <UCPTalarmPriority> property. When doing so, you must reference each priority level with the identifier listed in Table 23.

Each time an Alarm Generator generates an alarm, the *priority_level* field of the alarm data points chosen for the Alarm Generator will be updates to the priority level chosen here.

Table 23 Alarm Priority Levels

Identifier	Notes
PR_LEVEL_0	Lowest alarm priority level
PR_LEVEL_1	
PR_LEVEL_2	
PR_LEVEL_3	Highest alarm priority level
PR_1	Life Safety Fire Alarms
PR_2	Property Safety Fire Alarms
PR_3	Fire Supervisory Alarm
PR_4	Fire Trouble/Fault (Display)
PR_6	Fire Pre-Alarm, HVAC Critical Equipment Alarm
PR_8	HVAC Alarms (BACnet Priority 8)

Identifier	Notes
PR_10	HVAC Critical Equipment RTN, Fire RTN (Display)
PR_16	HVAC RTN (lowest priority)
PR_NUL	Value not available

7.2.1.2.2 *Comparison Functions*

Table 24 describes the comparison functions an Alarm Generator can use when comparing the values of the input and compare data points. You can select a comparison function for the Alarm Generator by filling in the <UCPTcompFunction> property. When doing so, you must reference each comparison function with the identifier strings listed in Table 24.

Table 24 Comparison Functions

Identifier	Description
`FN_GT`	Greater than. In this case, an alarm will be generated if the input value is greater than the compare value.
`FN_LT`	Less than. In this case, an alarm will be generated if the input value is less than the compare value.
`FN_GE`	Greater than or equal. In this case, an alarm will be generated if the input value is greater than or equal to the compare value.
`FN_LE`	Less than or equal. In this case, an alarm will be generated if the input value is less than or equal to the compare value.
`FN_EQ`	Equal. In this case, an alarm will be generated if the input value is equal to the compare value.
`FN_NE`	Not equal. In this case, an alarm will be generated if the input value is not equal to the compare value.
`FN_LIMIT`	Compare against the limits defined by the high and low limit offset fields. For more information on how these limits are used, see *Hysteresis Levels and Offset Limits* on page 7-13.

Different comparison functions should be used for different data point types, depending on the <UCPTbaseType> of the data point. Table 25 lists the different data point base types, and the comparison functions you can use with them.

Table 25 Base Types and Comparison Functions

Base Type	Valid <UCPTcompFunction>
BT_UNKNOWN, BT_ENUM, BT_ARRAY, BT_STRUCT, BT_UNION, BT_BITFIELD	FN_EQ, FN_NE
BT_SIGNED_CHAR, BT_UNSIGNED_CHAR, BT_SIGNED_SHORT, BT_UNSIGNED_SHORT, BT_SIGNED_LONG, BT_UNSIGNED_LONG, BT_FLOAT, BT_SIGNED_QUAD, BT_UNSIGNED_QUAD, BT_DOUBLE	FN_GT, FN_LT, FN_GE, FN_LE, FN_EQ, FN_NE, FN_LIMIT

You can make inequality comparisons between SNVT_switch (BT_STRUCT) data points, or between SNVT_lev_disc (BT_ENUM) data points. Table 26 lists the <UCPTcompFunction> identifiers you could use for these special comparisons. A description of how these comparisons are made follows Table 26.

Table 26 Exceptions to Base Types and Comparison Functions

SNVT	**Valid <UCPTcompFunction>**
SNVT_switch	FN_GT, FN_LT, FN_GE, FN_LE, FN_EQ, FN_NE
SNVT_lev_disc	FN_GT, FN_LT, FN_GE, FN_LE, FN_EQ, FN_NE

Comparisons made with SNVT_switch data points are enumeration-based comparisons based on the *value* field of the SNVT_switch. If the *value* field is between 0.5 and 100.0, the SNVT_switch is considered ON and that will be the basis of the comparison. If the *value* field is between 0.0 and 0.4, the SNVT_switch will be considered OFF. In this way you could compare SNVT_switch data points. For example, if the input data point was ON, the compare data point was OFF, and the comparison function selected was FN_GT, the comparison would return True because ON is considered greater than OFF.

This is also true for SNVT_lev_disc data points, which take five enumerations: OFF, LOW, MEDIUM, HIGH, and ON. If the input data point was LOW, the compare data point was HIGH and the comparison function was FN_GT, the function would return False, because LOW is not greater than HIGH.

7.2.1.2.3 Hysteresis Levels and Offset Limits

The four offset limit properties are named <UCPTlowLimit1Offset>, <UCPTlowLimit2Offset>, <UCPThighLimit1Offset>, and <UCPThighLimit2Offset>. The Alarm Generator will use these offsets to determine if an alarm condition exists when the <UCPTcompFunction> selected for the Alarm Generator is FN_LIMIT.

Table 27 lists the four offset limits, and the condition set that causes each one to generate an alarm. It also lists the status that the <UCPTpointStatus> of the input and alarm data points will be updated to when an alarm is generated based on each offset limit in the **Alarm Status** column.

Table 27 Hysteresis Levels and Offset Limits

Offset Limit	**Alarm Generated When....**	**Alarm Status**
<UCPThighLimit1Offset>	Input Value>Compare Value + UCPThighLimit1Offset	AL_HIGH_LMT_ALM1
<UCPThighLimit2Offset>	Input Value>Compare Value + UCPThighLimit2Offset	AL_HIGH_LMT_ALM2
<UCPTlowLimit1Offset>	Input Value<Compare Value – UCPTlowLimit1Offset	AL_LOW_LMT_ALM1
<UCPTlowLimit2Offset>	Input Value<Compare Value – UCPTlowLimit2Offset	AL_LOW_LMT_ALM2

Each time an alarm is generated based on any of these offset limits, the value of the input data point must return to a value inside the hysteresis range for that limit, and the time period specified by the <UCPTclrTime> property must elapse, before the alarm is cleared. Only then could another alarm be generated based on that offset limit.

The Alarm Generator's hysteresis levels determine the value the input data point must return to for each alarm condition to be cleared. Table 28 describes how these levels are calculated for each of the offset limits listed above.

Table 28 Alarm Generator Hysteresis Levels

Offset Limit Causing Alarm	Alarm Cleared When...
<UCPThighLimit1Offset>	Input Value<=Comp Value+ UCPThighLimit1Offset – SCPThysHigh1
<UCPThighLimit2Offset>	Input Value<=Comp Value+ UCPThighLimit2Offset – SCPThysHigh2
<UCPTlowLimit1Offset>	Input Value>= Compare Value – UCPTlowLimit1Offset + SCPThysLow1
<UCPTlowLimit2Offset>	Input Value>= Compare Value – UCPTlowLimit2Offset + SCPThysLow2

When an alarm is cleared, the data point is updated to the next lowest alarm level. For example, when an AL_LOW_LMT_ALM_2 alarm is cleared, the data point is updated to AL_LOW_LMT_ALM_1. When that condition clears, the data point is updated to AL_NO_CONDITION.

Table 29 describes this process in more detail.

Table 29 Alarm Statuses

Event	Input Data Point Status	Comments
Value of input data point is normal.	AL_NO_CONDITION	No alarm condition.
Value of input data point goes above first level (UCPThighLimit1Offset).	AL_HIGH_LMT_ALM1	Updated to the first alarm condition.
Value of input data point goes above second level (UCPThighLimit2Offset).	AL_HIGH_LMT_ALM2	Updated to the second, and more severe, alarm condition.
Value of input data point goes below hysteresis level for the second alarm condition.	AL_HIGH_LMT_ALM1	Updated back to the first alarm condition, as the data point has not yet reached the hysteresis level for that condition.
Value of input data point goes below hysterisis level for the first alarm condition.	AL_NO_CONDITION	Updated back to normal status.

The following diagram depicts the four different alarm conditions, as well as the corresponding hysteresis levels that must be reached to clear the alarms generated for each condition, in a line chart.

Please note that the diagram uses enumerations to label the hysteresis levels the input value must reach for each alarm status to be cleared. For example, AL_HIGH_LMT_CLR_2 represents the value necessary to clear the AL_HIGH_LMT_ALM_2 alarm status. AL_HIGH_LMT_CLR_1 represents the value necessary to clear the AL_HIGH_LMT_ALM_1 alarm status. The data points in your network will not be updated to these statuses at any time.

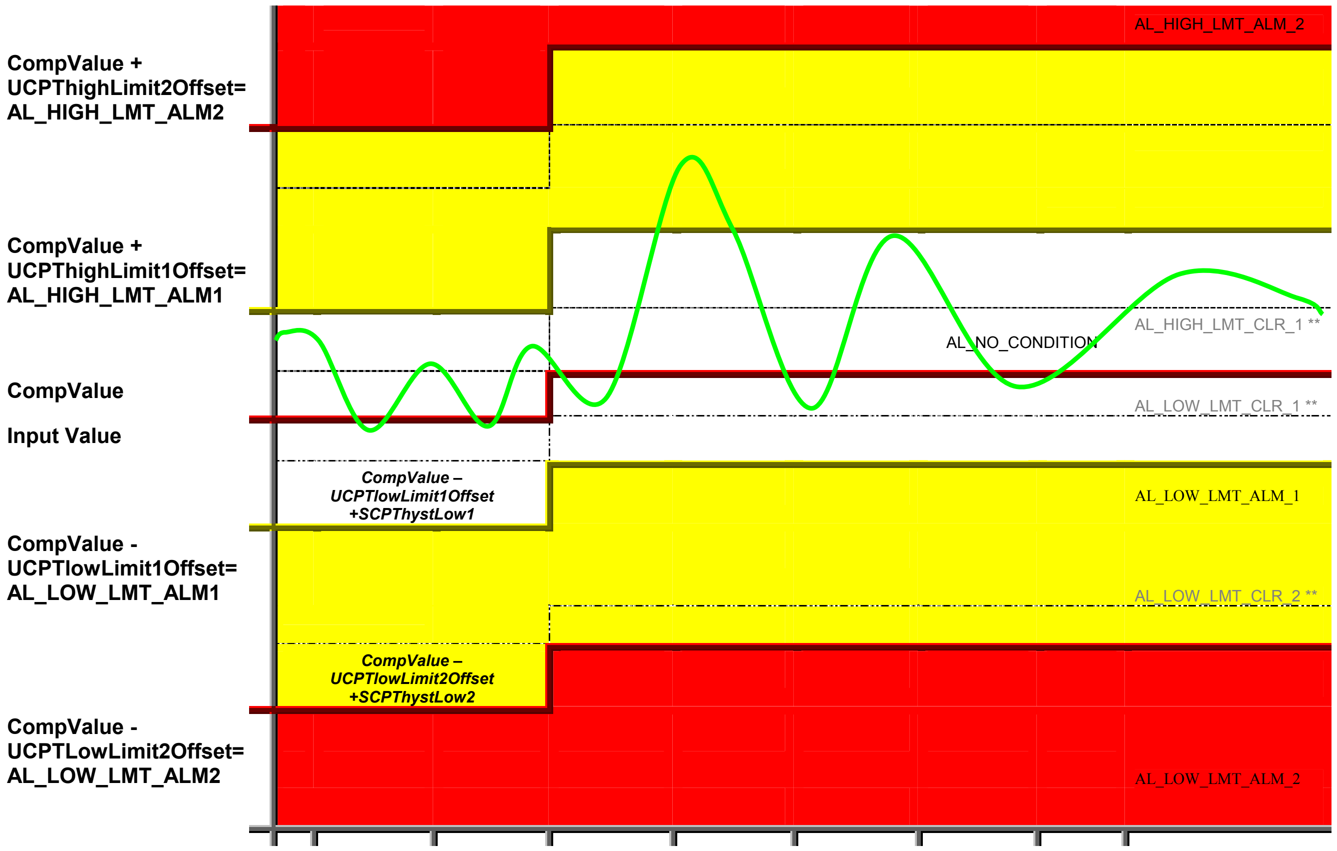


Figure 1 Hysteresis Levels and Offset Limits

7.2.1.3 AlarmGeneratorSet

You can use the AlarmGeneratorSet function to create a new Alarm Generator, or to overwrite the configuration of an existing Alarm Generator. You can create up to 40 Alarm Generators per *i.*LON 100.

The Alarm Generators to be created or written to by the function are signified by a list of <Alarm> elements in the <Data> parameter. The properties that you must define within each <Alarm> element are the same, whether you are creating a new Alarm Generator or modifying an existing Alarm Generator. The previous section, *AlarmGeneratorGet,* describes these properties.

NOTE: When modifying an existing Alarm Generator, any optional properties such as <AlarmDataPoint> and <Alarm2DataPoint> that are left out of the input string will be erased. Old values will not be carried over, so you must fill in every property when writing to an Alarm Generator, even if you are not changing all of the values.

The first invocation of the AlarmGeneratorSet function will generate the AlarmGenerator.XML file in the /root/Config/software directory of the *i.*LON 100, if the file does not already exist.

When creating or modifying an Alarm Generator with AlarmGeneratorSet, you may want to use output from AlarmGeneratorGet as the basis for your <Data> parameter. You would then only need to modify the values of each property to match the new configuration you want, as opposed to re-creating an entire string like the one shown below.

The following example uses the AlarmGeneratorSet function to create a new Alarm Generator that uses a data point called NVL_DataValueAG1 as its input data point.

<Data> Parameter

```
<Data>
  <iLONAlarmGenerator>
        <Alarm>
             <UCPTindex></UCPTindex>
             <UCPTdescription>Alarm Generator 1</UCPTdescription>
             <UCPTfbName></UCPTfbName>
             <SCPTalrmIhbT>0 00:00:00.000</SCPTalrmIhbT>
             <UCPTalarmPriority>PR_LEVEL_1</UCPTalarmPriority>
             <UCPTpollOnResetDelay>0.0</UCPTpollOnResetDelay>
             <UCPTpollRate>0.0</UCPTpollRate>
             <UCPTalarm2Description>none</UCPTalarm2Description>
             <InputDataPoint>
                    <UCPTpointName>NVL_DataValueAG1</UCPTpointName>
             </InputDataPoint>
             <CompareDataPoint>
                    <UCPTpointName>NVL_CompareValueAG1</UCPTpointName>
             </CompareDataPoint>
             <AlarmDataPoint>
                    <UCPTpointName>NVL_AlarmGenOut1</UCPTpointName>
             </AlarmDataPoint>
             <Alarm2DataPoint>
                    <UCPTpointName></UCPTpointName>
             </Alarm2DataPoint>
             <UCPTcompFunction>FN_LIMIT</UCPTcompFunction>
             <UCPTalarmSetTime>0 00:00:00.000</UCPTalarmSetTime>
             <UCPTalarmClrTime>0 00:00:00.000</UCPTalarmClrTime>
             <UCPTlowLimit1Offset>5.0</UCPTlowLimit1Offset>
             <UCPTlowLimit2Offset>5.0</UCPTlowLimit2Offset>
             <UCPThighLimit1Offset>5.0</UCPThighLimit1Offset>
             <UCPThighLimit2Offset>5.0</UCPThighLimit2Offset>
```

```
              <SCPThystHigh1>50.00</SCPThystHigh1>
              <SCPThystHigh2>75.00</SCPThystHigh2>
              <SCPThystLow1>50.00</SCPThystLow1>
              <SCPThystLow2>75.00</SCPThystLow2>
        </Alarm>
   </iLONAlarmGenerator>
 </Data>
```

<Result> Parameter

```
<Result>
    <iLONAlarmGenerator>
      <Alarm>
            <UCPTindex>2</UCPTindex>
        </Alarm>
    </iLONAlarmGenerator>
</Result>
```

7.2.1.4 AlarmGeneratorDelete

You can use the AlarmGeneratorDelete function to delete an Alarm Generator. You must reference the Alarm Generator to be deleted by its index number in the <Data> parameter you supply to the function, as in the example below.

<Data> Parameter

```
<Data>
  <iLONAlarmGenerator>
    <Alarm>
        <UCPTindex>1</UCPTindex>
    </Alarm>
  </iLONAlarmGenerator>
</Data>
```

<Result> Parameter

```
<Result>
   <iLONAlarmGenerator>
       <Alarm>
              <UCPTindex>1</UCPTindex>
       </Alarm>
   </iLONAlarmGenerator>
</Result>
```

8 Alarm Notifier

Use the Alarm Notifier application to log user-defined alarm conditons, and to generate e-mail messages and data point updates each time an alarm condition occurs. This section provides an overview of how Alarm Notifiers work, including how you can define alarm conditions and program your Alarm Notifiers to respond to them.

User-Defined Alarm Conditions

When you create an Alarm Notifier, you will specify a group of input data points. The Alarm Notifier will read the status of these data points each time they are updated to determine if they have reached alarm conditions. The statuses that the Alarm Notfiier will consider alarm conditions are completely user-defined. You will define these conditions by creating active and passive alarm conditon sets for the Alarm Notifier.

For each condition set you create, you will select an alarm type (active or passive) and a point status. Each time an input data point is updated and its <UCPTpointStatus> matches that status, an alarm notification will occur. If it is generated based on a status assigned to an active alarm conditon set, it is considered an active alarm. If it is generated based on a staus assigned to a passive condition set, it is considered a passive alarm. You can create as many active and passive alarm condition sets as you like per Alarm Notifier.

There are several scenarios you could consider when creating Alarm Notifiers. For example, you could set up Alarm Notifiers to generate alarm notifications based on the statuses of the data points updated by your Alarm Generators. For more information on Alarm Generators, see Chapter 7, *Alarm Generator*.

You may also recall from Chapter 6 that some data points exist in the Data Server to monitor the amount of memory that a Data Logger's log file has consumed. You could set up an Alarm Notfier to generate alarm notifications when a log file becomes full.

Alarm Destinations

You will create destination sets for your Alarm Notifiers. These destination sets determine how the Alarm Notifier will respond when an alarm occurs.

For each destination set, you can specify an output data point. This data point will be updated each time an alarm notification occurs and uses that particular destination. You can also specify an e-mail profile for each destination. The e-mail profile will cause an e-mail to be sent to an address of your choice each time the destination is used. The next section provides more information on e-mail profiles.

You can create as many active and passive destination sets as you like per Alarm Notifier. The passive destination will be used when a passive alarm notification occurs, and the active destinations are used when an active alarm notification occurs.

E-mail Profiles

You can create e-mail profiles and assign these profiles to the destination sets you have created for your Alarm Notifier. Each e-mail profile contains an e-mail address. When a destination using an e-mail profile is used, an e-mail will be sent to the address defined for that profile.

You can specify the message text, subject heading, and attachment to be included with each e-mail. E-mail 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 might occur on your network.

Auto-Generated Log Files

Each Alarm Notifier will generate its own log file. It will add an entry to this log file each time it generates an alarm notification. You can find these log files in the /root/AlarmLog directory of the *i.*LON 100. 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.

In addition, the Alarm Notifier application generates a summary log that summarizes the log entries made by all the 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.

You can create the log files in either a text format (.csv) or binary format (.dat). You will establish this when you create your Alarm Notifiers. You can read these log files by opening the log files via an FTP session, or by using the AlarmNotifierRead function. You can use the Alarm NotifierWrite function to acknowklede and comment on the alarm notifications stored in the log files.

8.1 AlarmNotifier.XML

The AlarmNotifier.XML file stores the configuration of the Alarm Notifiers that you have added to the *i.*LON 100. You can create up to 40 Alarm Notifiers per *i.*LON 100. Each Alarm Notifier is signified by an <Alarm> element in the XML file.

You can create Alarm Notifiers with the AlarmNotifierSet SOAP function, or by manually editing the AlarmNotifier.XML file and downloading it to the *i.*LON 100 via FTP. The sections following this example provide instructions and guidelines to assist you when doing so.

The following represents a sample AlarmNotifier.XML file for an *i.*LON 100 with one defined alarm Notifier. This Alarm Notifier generates alarm notifications based on the status of the data point NVL_nvoLevelAlarm. This data point monitors the log level of a Data Logger. As you may recall from Chapter 6, this data point will be set to the alarm condition AL_ALM_CONDITION when the volume of the Data Logger reaches its pre-defined log level.

The Alarm Notifier defined by the example below triggers an alarm notification when this occurs, and updates the value of the NVL_nviDlClear data point to 100.0 1. The update to NVL_nviDlClear will clear out the Data Logger's log file. So, the Alarm Notifier defined by the XML file below monitors the log level of a Data Logger, and empties the Data Logger's log file when it becomes full.

```
<?xml version="1.0" ?>
  <iLONAlarmNotifier>
    <SCPTobjMajVer>1</SCPTobjMajVer>
    <SCPTobjMinVer>1</SCPTobjMinVer>
    <Alarm>
      <UCPTindex>11</UCPTindex>
      <UCPTlastUpdate>2002-07-04T15:08:54Z</UCPTlastUpdate>
      <UCPTdescription>Monitors the log level of a Data
       Logger </UCPTdescription>
      <UCPTfbName>Alarm Notifier- 1</UCPTfbName>
      <SCPTdelayTime>0.0</SCPTdelayTime>
      <UCPTsumLogSize>100</UCPTsumLogSize>
      <UCPThistLogSize>100</UCPThistLogSize>
      <UCPTlogFormat>LF_BINARY</UCPTlogFormat>
      <UCPTemailAggregTime>10</UCPTemailAggregTime>
      <Point>
        <UCPTindex>0</UCPTindex>
        <UCPTpointName>NVL_nvoLevelAlarm</UCPTpointName>
        <UCPTalarmFlags>0 1 0 0 0 0 0</UCPTalarmFlags>
        <UCPTalarmGroup>1</UCPTalarmGroup>
        <UCPTpriority>7</UCPTpriority>
        <UCPTdescription>log level of data logger 1
         </UCPTdescription>
      </Point>
      <Mail>
        <UCPTindex>0</UCPTindex>
        <UCPTemailNickName>Joerg</UCPTemailNickName>
        <UCPTemailAddress>js@nova</UCPTemailAddress>
        <UCPTemailFormat>occured %dy/%dm/%dd </UCPTemailFormat>
        <UCPTemailSubject>Nofifier1: %ad</UCPTemailSubject>
        <UCPTemailAttachment>/root/Data/log1.csv</UCPTemailAttachment>
      </Mail>
      <ActiveAlarm>
        <UCPTindex>0</UCPTindex>
        <UCPTlevel>1</UCPTlevel>
        <UCPTalarmText>Log 30 percent full</UCPTalarmText>
        <UCPTactAlarmType>AL_ALM_CONDITION</UCPTactAlarmType>
      </ActiveAlarm>
      <PassiveAlarm>
        <UCPTindex>0</UCPTindex>
        <UCPTlevel>255</UCPTlevel>
        <UCPTalarmText>Normal Condition</UCPTalarmText>
        <UCPTpasAlarmType>AL_NO_CONDITION</UCPTpasAlarmType>
      </PassiveAlarm>
      <AlarmDest>
        <UCPTindex>0</UCPTindex>
        <UCPTdestEnable />
        <ActiveDest>
           <UCPTindex>0</UCPTindex>
           <UCPTemailNickName>Joerg</UCPTemailNickName>
           <UCPTpointName>NVL_nvoDlClear</UCPTpointName>
           <UCPTpointValue>100.0 1</UCPTpointValue>
           <UCPTminLevel>2</UCPTminLevel>
           <UCPTmaxLevel>0</UCPTmaxLevel>
           <UCPTnackDelay>0</UCPTnackDelay>
        </ActiveDest>
        <PassiveDest>
           <UCPTindex>0</UCPTindex>
           <UCPTemailNickName>Joerg</UCPTemailNickName>
           <UCPTpointName>NVL_nvoDlClear</UCPTpointName>
           <UCPTpointValue>0.0 0</UCPTpointValue>
        </PassiveDest>
      </AlarmDest>
   </Alarm>
 </iLONAlarmNotifier>
```

8.2 Creating and Modifying the AlarmNotifier.XML File

You can create and manage the AlarmNotifier.XML file with the AlarmNotifierSet SOAP function. The following section, *Alarm Notifier SOAP Interface*, describes how to use AlarmNotifierSet and the other SOAP functions provided for the Alarm Notifier application.

Alternatively, you can create and manage the AlarmNotifier.XML file manually with an XML editor and download it to the *i.*LON 100 via FTP. Echelon does not recommend this, as the *i.*LON 100 will require a reboot to read the configuration of the downloaded file. Additionally, the *i.*LON 100 performs error checking on all SOAP messages it receives before writing to the XML file. It will not perform error checking on any XML files you download via FTP, and thus the application may not boot properly.

If you plan to create the XML file manually, you should review the rest of this chapter first, as it describes the elements and properties in the XML file that define each Alarm Notifier's configuration. For instructions on creating or modifying an XML file manually, see *Manually Modifying an XML Configuration File* on page 15-1.

8.2.1 Alarm Notifier SOAP Interface

The SOAP interface for the Alarm Notifier application includes seven functions. Table 30 lists and describes these functions. For more information on each function, see the sections following Table 30.

Table 30 Alarm Notifier SOAP Functions

Function	Description
AlarmNotifierList	Use this function to generate a list of the Alarm Notifiers that you have added to the *i.*LON 100. For more information, see *AlarmNotifierList* on page 8-5.
AlarmNotifierGet	Use this function to return the configuration of an Alarm Notifier. For more information, see *AlarmNotifierGet* on page 8-6.
AlarmNotifierSet	Use this function to create an Alarm Notifier, or to overwrite the configuration of an exisiting Alarm Notifier. For more information, see *AlarmNotifierSet* on page 8-19.
AlarmNotifierRead	Each time an Alarm Notifier causes an alarm notification, it will record a log entry for that notification. Use this function to retrieve some or all of the log entries that an Alarm Notifier has recorded. For more information, see *AlarmNotifierRead* on page 8-20.
AlarmNotifierWrite	Use this function to acknowledge an alarm notification, or group of alarm log notifications, made by the Alarm Notifier. You optionally insert comments into the log entry for each alarm notification with this function. For more information, see *AlarmNotifierWrite* on page 8-26.
AlarmNotifierClear	Use this function to clear log entries from an Alarm Notifier's log file. For more information, see *AlarmNotifierClear* on page 8-29.
AlarmNotifierDelete	Use this function to delete an Alarm Notifier. For more information, see *AlarmNotifierDelete* on page 8-31.

8.2.1.1 AlarmNotifierList

Use the AlarmNotifierList function to retrieve a list of the Alarm Notifiers that you have added to the *i.*LON 100. The AlarmNotifierList function takes an empty string as its <Data> parameter, as in the example below.

The function returns the major and minor version numbers of the firmware implementation the Alarm Notifier application is using in the <Result> parameter. The <Result> parameter also includes an <Alarm> element for each Alarm Notifier that you have added to the *i.*LON 100. The next section, *AlarmNotifierGet,* describes the properties that are included in each of these elements.

You could use the list of <Alarm> elements returned by this function as input for the AlarmNotifierGet function. The AlarmNotifierGet function would then return the configuration of each Alarm Notifier included in the list.

<Data> Parameter

```
Empty String
```

<Result> Parameter

```
<Result>
 <iLONAlarmNotifier>
  <SCPTobjMajVer>1</SCPTobjMajVer>
  <SCPTobjMinVer>1</SCPTobjMinVer>
  <Alarm>
     <UCPTindex>0</UCPTindex>
     <UCPTlastUpdate>2002-06-20T12:37:07Z</UCPTlastUpdate>
     <UCPTdescription>Control of the Data Logger</UCPTdescription>
     <UCPTfbName>Alarm Notifier- 0</UCPTfbName>
  </Alarm>
  <Alarm>
     <UCPTindex>1</UCPTindex>
     <UCPTlastUpdate>200206-20T12:37:07Z</UCPTlastUpdate>
     <UCPTdescription>Temperature alarm</UCPTdescription>
     <UCPTfbName>Alarm Notifier- 1</UCPTfbName>
  </Alarm>
 </iLONAlarmNotifier>
</Result>
```

8.2.1.2 AlarmNotifierGet

You can use the AlarmNotifierGet function to return the configuration of any Alarm Notifier that you have added to the *i.*LON 100. You must reference the Alarm Notifier whose configuration is to be returned by its index number in the <Data> parameter you supply to the function, as in the example below.

<Data> Parameter

```
<Data>
  <iLONAlarmNotifier>
    <Alarm>
        <UCPTindex>0</UCPTindex>
    </Alarm>
  </iLONAlarmNotifier>
</Data>
```

<Result> Parameter

```
<Result>
  <iLONAlarmNotifier>
    <Alarm>
       <UCPTindex>0</UCPTindex>
       <UCPTlastUpdate>2002-06-21T07:59:23Z</UCPTlastUpdate>
       <UCPTdescription>Temperature Sensor Device</UCPTdescription>
       <UCPTfbName>Alarm Notifier- 0</UCPTfbName>
       <SCPTdelayTime>0.0</SCPTdelayTime>
       <UCPTsumLogSize>100</UCPTsumLogSize>
       <UCPThistLogSize>100</UCPThistLogSize>
       <UCPTlogFormat>LF_TEXT</UCPTlogFormat>
       <UCPTemailAggregTime>10</UCPTemailAggregTime>
       <Point>
          <UCPTindex>0</UCPTindex>
          <UCPTpointName>NVL_AlarmGenIn1</UCPTpointName>
          <UCPTalarmFlags>0 1 0 0 0 0 0</UCPTalarmFlags>
          <UCPTalarmGroup>0</UCPTalarmGroup>
          <UCPTpriority>1</UCPTpriority>
          <UCPTdescription></UCPTdescription>
       </Point>
        <Mail>
          <UCPTindex>0</UCPTindex>
          <UCPTemailNickName>Joerg</UCPTemailNickName>
          <UCPTemailAddress>js@nova</UCPTemailAddress>
          <UCPTemailFormat>occured%dy/%dm/%dd</UCPTemailFormat>
          <UCPTemailSubject>Alarm Nofifier1: %ad</UCPTemailSubject>
          <UCPTemailAttachment>/root/Data/log1.csv</UCPTemailAttachment>
       </Mail>
       <ActiveAlarm>
          <UCPTindex>0</UCPTindex>
          <UCPTlevel>1</UCPTlevel>
          <UCPTalarmText>Alarm (Binary)</UCPTalarmText>
          <UCPTactAlarmType>AL_ALM_CONDITION</UCPTactAlarmType>
        </ActiveAlarm>
        <ActiveAlarm>
          <UCPTindex>1</UCPTindex>
          <UCPTlevel>2</UCPTlevel>
          <UCPTalarmText>Alarm (Binary) offline</UCPTalarmText>
          <UCPTactAlarmType>AL_OFFLINE</UCPTactAlarmType>
        </ActiveAlarm>
        <PassiveAlarm>
          <UCPTindex>0</UCPTindex>
          <UCPTlevel>255</UCPTlevel>
          <UCPTalarmText>Normal Condition</UCPTalarmText>
```

```
                <UCPTpasAlarmType>AL_NO_CONDITION</UCPTpasAlarmType>
                <UCPTpasAlarmType>AL_NUL</UCPTpasAlarmType>
              </PassiveAlarm>
              <AlarmDest>
                <UCPTindex>0</UCPTindex>
                <UCPTdestEnable>NVL_nviWeekday</UCPTdestEnable>
                <ActiveDest>
                  <UCPTindex>0</UCPTindex>
                  <UCPTpointName>NVL_nvoAlarmFlag1</UCPTpointName>
                  <UCPTpointValue>77.0 1</UCPTpointValue>
                  <UCPTminLevel>1</UCPTminLevel>
                  <UCPTmaxLevel>0</UCPTmaxLevel>
                  <UCPTnackDelay>0</UCPTnackDelay>
                </ActiveDest>
                <PassiveDest>
                   <UCPTindex>1</UCPTindex>
                   <UCPTpointName>NVL_nvoAlarmFlag1</UCPTpointName>
                   <UCPTpointValue>MEDIUM</UCPTpointValue>
                </PassiveDest>
              </AlarmDest>
              <AlarmDest>
                   <UCPTindex>1</UCPTindex>
                   <UCPTdestEnable>NVL_nviWeekend</UCPTdestEnable>
                   <ActiveDest>
                     <UCPTindex>0</UCPTindex>
                     <UCPTpointName>NVL_nvoAlarmFlag1</UCPTpointName>
                     <UCPTemailNickName>Joerg</UCPTemailNickName>
                     <UCPTpointValue>100.0 1</UCPTpointValue>
                     <UCPTminLevel>2</UCPTminLevel>
                     <UCPTmaxLevel>0</UCPTmaxLevel>
                     <UCPTnackDelay>0</UCPTnackDelay>
                   </ActiveDest>
                   <PassiveDest>
                     <UCPTindex>1</UCPTindex>
                     <UCPTpointName>NVL_nvoAlarmFlag1</UCPTpointName>
                     <UCPTpointValue>Off Value</UCPTpointValue>
                     <UCPTminLevel>255</UCPTminLevel>
                     <UCPTmaxLevel>255</UCPTmaxLevel>
                   </PassiveDest>
               </AlarmDest>
           </Alarm>
        </iLONAlarmNotifier>
      </Result>
```

The function returns an <Alarm> element for each Alarm Notifier referenced in the <Data> parameter you supplied to the function. The properties included in each element are initially defined when the Alarm Notifier is created. You can write to them using the AlarmNotifierSet function. Table 31 describes these properties.

For more information on the AlarmNotifierSet function, see *AlarmNotifierSet* on page 8-19.

Table 31 AlarmNotifierGet Output Properties

Property	Description

Property	Description
`<UCPTindex>`	The index number assigned to the Alarm Notifier must be in the range 0-32,767. As mentioned earlier, you can use the AlarmNotifierSet function to create a new Alarm Notifier, or to modify an existing Alarm Notifier. If you do not specify an index number in the <Data> parameter you supply to AlarmNotifierSet, the function will create a new Alarm Notifier using the first available index number. If you specify an index number that is already being used, the function will overwrite the configuration of the Alarm Notifier using that index number with the settings defined in the <Data> parameter.
`<UCPTlastUpdate>`	A timestamp indicating the last time the configuration of the Alarm Notifier was updated. This timestamp uses the following format: YYYY-MM-DD**T**HH:MM:SS**Z** The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Alarm Notifier was last updated. The second segment (**T**HH:MM:SS) represents the time of day the configuration of the Alarm Notifier was last updated, in UTC (Coordinated Universal Time). UTC is the current term for what was commonly referred to as Greenwich Meridian Time (GMT). Zero (0) hours UTC is midnight in Greenwich England, which lies on the zero longitudinal meridian. Universal time is based on a 24 hour clock, therefore, an afternoon hour such as 4 pm UTC would expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002.
`<UCPTdescription>`	A user-defined description of the Alarm Notifier. This can be a maximum of 228 characters.
`<UCPTfbName>`	The functional block name assigned to the Alarm Notifier in LONMAKER. You can write to this property, but each time you use the *i.*LON 100 Configuration Software to view the Alarm Notifier, it will be reset to match the functional block name defined in LONMAKER.
`<SCPTdelayTime>`	The minimum time (in seconds) that must pass after this Alarm Notifier has logged an alarm notification before the e-mail profiles for the Alarm Notifier can be used, or the output data points for this Alarm Notifier can be updated. This property defaults to 0.
`<UCPTsumLogSize>`	The size of the summary alarm log file, in kilobytes. The summary alarm log includes records for all current acknowledged and unacknowledged alarms. Please note that the total size of the log files for all Alarm Notifiers (and Data Loggers) on the *i.*LON 100 can not exceed the size of the flash memory stored in the *i.*LON 100. The *i.*LON 100 will stop writing to the log files when it only has 256 Kb of flash memory remaining.

Property	Description
`<UCPThistLogSize>`	The size of the historical alarm log file, in kilobytes. The historical alarm log contains a record for any acknowledged alarm. Each record includes the description, acknowledgment time and comment entered for the alarm. Please note that the total size of the log files for all Alarm Notifiers (and Data Loggers) on the *i.*LON 100 can not exceed the size of the flash memory stored in the *i.*LON 100. The *i.*LON 100 will stop writing to the log files when it only has 256 Kb of flash memory remaining.
`<UCPTlogFormat>`	Either LF_BINARY or LF_TEXT. This property determines whether the log file will be generated as a binary file, or as a text file.
`<UCPTemailAggregTime>`	The time, in milliseconds, to wait after an alarm occurs before using the email profiles defined for the Alarm Notifier. This may be useful if you want to prevent multiple e-mails from being sent to the same address at the same time. The default value used if you do not define this property is 0. The maximum value is 65,535 milliseconds. **NOTE:** The <UCPTemailAggregTime> counter resets every time an alarm occurs. Therefore, if multiple alarms occur before the aggregation period expires, the emails for those alarms will be merged and sent as a single email notification. The *i.*LON 100 will send the email automatically after 100 alarms have been merged. This may be useful if multiple alarms occur within a few moments of each other, but you should take it into consideration before setting this property to a high value.
`<Point>`	An alarm notification will occur each time any of the input data points defined for an Alarm Notifier are updated, and the data point's <UCPTpointStatus> matches the status defined for any of the Alarm Notifier's active or passive alarm condition sets. You can specify as many input data points as you like per Alarm Notifier. The input data points for an Alarm Notifier are signified by a list of <Point> elements in the <Data> parameter. For a description of the properties that must be defined within each <Point> element, see *Input Data Points* on page 8-10. You can specify as many input data points as you want for each Alarm Notifier.
`<Mail>`	An e-mail profile contains an e-mail address, message text, subject heading, and an attachment file. An e-mail message with the subject heading, message text and attachment will be sent to the address provided each time the e-mail profile is used. You will reference these e-mail profiles when you set up the Alarm Notifier's active and passive alarm destination sets. You can create as many e-mail profiles as you want for each Alarm Notifier, but each alarm destination can reference only one e-mail profile. The e-mail profiles for an Alarm Notifier are signified by a list of <Mail> elements. For a description of the properties that must be defined within each <Mail> element, see *E-mail Profiles* on page 8-13.

Property	Description
`<ActiveAlarm>`	If the input data point is updated and matches the conditions defined by any of the active alarm condition sets, it is considered an active alarm. In this case, the Alarm Notifier will use its active destinations. You can create as many active alarm condition sets as you want per Alarm Notifier. The active alarm condition sets for an Alarm Notifier are signified by a list of <ActiveAlarm> elements. For a description of the properties that must be defined within each <ActiveAlarm> element, see *Active and Passive Alarm Condition Sets* on page 8-15.
`<PassiveAlarm>`	If the input data point is updated and matches the conditions defined by any of the passive alarm condition sets, it is considered a passive alarm. In this case, the Alarm Notifier will use its passive destinations. You can create as many passive alarm condition sets as you want per Alarm Notifier. The passive alarm condition sets for an Alarm Notifier are signified by a list of <PassiveAlarm> elements. For a description of the properties that must be defined within each <PassiveAlarm> element, see *Active and Passive Alarm Condition Sets* on page 8-15.
`<AlarmDest>`	Each <AlarmDest> element defines a group of active and passive alarm destinations the Alarm Notifier will use. The active destinations are signified by a list of <ActiveDest> child elements within the <AlarmDest> element. The passive destinations are signified by a list of <PassiveDest> child elements within the <AlarmDest> element. For a description of the properties that must be defined within each of these child elements, see *Active and Passive Alarm Destinations* on page 8-16. Each <AlarmDest> element also contains 2 global elements: its index number (UCPTindex), and its enable data point (UCPTdestEnable). The <UCPTdestEnable> property is optional. You can reference a SNVT_Switch data point by its name (UCPTpointName) here. The <AlarmDest> will then be enabled when that data point is set to 100.0 1, or disabled if that data point is set to 0.0 0. You could set this data point with a LONWORKS switch, or with the Event Scheduler application. This allows you to enable or disable an Alarm Notifier's destination sets under different circumstances.

8.2.1.2.1 *Input Data Points*

The following table describes the properties that you must define within each <Point> element. As described in the previous section, each <Point> element defines an input data point for the Alarm Notifier. Each time any of the input data points are updated, the Alarm Notifier will check if it has reached an alarm condition.

If an input data point is updated and meets an active or passive alarm condition, then an alarm notification will be loggeed, and the applicable passive or active alarm destinations will be used.

Table 32 Input Data Point Properties

Property	Description
`<UCPTindex>`	The index number to be used within the Alarm Generator application for this data point. This does not have to match the index number assigned to the data point in the *i.*LON 100 Data Server.
`<UCPTpointName>`	The name of the data point, as defined in the *i.*LON 100 Data Server.

Property	Description
`<UCPTalarmFlags>`	This field is a series of seven Boolean values (0,1) separated by spaces that determine what information will be stored in the Alarm History and Alarm Summary Logs for this data point. The meanings of each byte in the string are described below: **Byte 1 (log_enable)**: When this byte is set to 0, each new alarm will be recorded in the Alarm History Log when it initially occurs. No further entries will be recorded into the log for the alarm. When this byte is set to 1, each new alarm will be recorded in the Alarm History Log when it initially occurs, and additional entries in the Alarm History Log will be added each time the status of the alarm changes. For example, an additional entry would be added for an alarm when it is acknowledged or cleared. **Byte 2 (invisible)**: When this byte is set to 0, alarm notifications for this data point will be recorded in the Alarm Summary Log. When this byte is set to 1, log entries for the data point will not be recorded in the Alarm Summary Log. **Byte 3 (reserved)**: Set this byte to 0. **Byte 4 (clear_required)**: When this byte is set to 0, the log entries for this data point will be automatically removed from the Alarm Summary Log when the alarm associated with the entry is acknowledged, or the alarm changes to a passive condition. You can acknowledge an alarm with the AlarmNotifierWrite function. When this byte is set to 1, you will need to clear all log entries from the Alarm Summary Log manually with the AlarmNotifierWrite function. Note that Byte 7 must be set to 0 to record log entries for the data point into the summary log. **Byte 5 (ack_required)**: When this byte is set to 1, all log entries made by the Alarm Notifier for this data point must be manually acknowlegded with the AlarmNotifierWrite function. When this byte is set to 0, each alarm triggered by the Alarm Notifier for this data point will be automatically acknowledged. In this case, they will not be recorded in the Alarm Summary Log if Byte 4 is also set to 0. Note that Byte 7 must be set to 0 to record log entries for the data point into the summary log. **Byte 6 (status_enabled)**: Not used. Set this byte to 0. **Byte 7 (cov_enabled)**: When this byte is set to 0, log entries for all changes in the alarm status this data point will be stored in the Alarm Summary Log. When this byte is set to 1, only the most recent change in the data point's alarm status will be logged by the Alarm Notifier in the Alarm Summary Log. Byte 3 and byte 6 should be left at 0. All other bytes can be set to ON (1) or OFF (0) by writing to the property. For example, to turn all fields on (except for reserved Bytes 3 and 6), enter the following string: `<UCPTalarmFlags>1 1 0 1 1 0 1</UCPTalarmFlags>` The default for this property is 0 0 0 0 0 0 0.

Property	Description
`<UCPTalarmGroup>`	The group number for alarm notifications caused by this data point. You can use group numbers to categorize alarms. Alarm groups can be numbered from 1 to 127.
`<UCPTpriority>`	The priority level to be assigned to the data point when it reaches an alarm condition. This must be an integer between 0 (high priority) and 255 (low priority). You can use priority levels to sort the alarms with the summary log view, or with the *i.*LON 100 Web pages. The default value is 0. For more information on priority levels and how you can use them, see *Data Point Values and Priority Levels* on page 3-5.
`<UCPTdescription>`	A user-defined description of the alarm condition for this data point. This can be a maximum of 201 characters long.

8.2.1.2.2 E-mail Profiles

Table 33 describes the properties that you must define within each <Mail> element. As described previously in this chapter, each <Mail> element defines an e-mail profile for the Alarm Notifier.

You will reference these e-mail profiles when creating the active and passive destination sets for your Alarm Notifier. An e-mail will be sent to the e-mail address specified for the profile each time any of the destinations that reference the profile are used. For more information on the active and passive alarm destination sets, see *Active and Passive Alarm Destinations* on page 8-16.

Table 33 E-mail Profile Properties

Property	Description
`<UCPTindex>`	The index number of the e-mail profile.
`<UCPTemailNickName>`	The name of the e-mail profile. You will use this name to reference the e-mail profile when setting up active and passive alarm destinations. It can be a maximum of 31 characters long.
`<UCPTemailAddress>`	The e-mail address for this profile. An e-mail will be sent to this address each time the profile is used. The address can be a maximum of 1024 characters long.
`<UCPTemailFormat>`	The message text e-mails sent by this profile will contain, as a string. The SOAP interface provides a group of macro arguments that can be used to automatically insert information about the alarm into the message. For example: %al occurred at %dy / %dm/ %dd %pn and reached the level of %va. For a description of the macro arguments you can use, see Table 34. This message can be a maximum of 4096 characters long.
`<UCPTemailSubject>`	The subject of the e-mails sent for this profile. This can be a maximum of 1024 characters long.
`<UCPTemailAttachment>`	The path of the attachment file that will be sent with the e-mails this profile sends. This must be an *i.*LON 100-based path. For example: /root/Data/log1.csv.

Property	Description
	The path can be a maximum of 1024 characters long.

Table 34 lists the macro arguments you can use to fill in the <UCPTemail Format> property within each mail element.

Table 34 Macro Arguments

Macro Argument	Description
%al	Alarm type. This is the current status (UCPTpointStatus) of the data point that caused the alarm.
%at	Alarm type number. This is the integer value that maps to the point status (UCPTpointStatus) that defines the alarm type.
%dm	The month the alarm occurred, as an integer between 1 and 12.
%dd	The day the alarm occurred, as an integer between 1 and 31.
%dy	The year the alarm occurred, as a 4-digit integer, e.g. 1997.
%dt	The date the alarm occurred, expressed in the following format: YYYY-MM-DD For example: 2002-30-10
%gr	Alarm group number. This is determined by the <UCPTalarmGroup> property assigned to the data point that caused the alarm within the Alarm Notifier.
%lm	Alarm limit. This is the value limit the input data point exceeded to be updated to its current alarm status by the Alarm Generator application. If no Alarm Generator is being used with the input data point, this will return 0.
%ls	Alarm location string. This is the text stored in the <UCPTlocation> property of the data point that caused the alarm.
%ob	The index number assigned to the data point that caused the alarm in the *i.*LON 100 Data Server.
%pr	The priority of the alarm.
%ps	Percent sign ("%").
%s1	One second delay for paging strings. This causes a one second delay in the writing of the e-mail.
%si	SNVT ID of the data point that caused the alarm.
%t1	The hour the alarm occurred, in 12-hour format. For example, this would return 10 for an alarm that occurred at 10:00 AM or 10:00 PM.
%t2	The hour the alarm occurred, in 24-hour format. For example, this would return 16 for an alarm that occurred at 4 PM.
%ta	Returns "A" for alarms that occurred in the morning, or "P" for alarms that occurred in the afternoon.

Macro Argument	Description
%tm	The minute that the alarm occurred.
%ts	The second that the alarm occurred.
%th	The millisecond that the alarm occurred.
%ti	The time that the alarm occurred, expressed in the following format: HH:MM:SS For example, 08:12:22 indicates an alarm time of 8 AM, 12 minutes and 22 seconds.
%va	The current value of the data point that caused the alarm.
%ad	The alarm description. This is taken from the alarm description (UCPTdescription) defined for the data point that caused the alarm in the Alarm Notifier.
%pn	The name of the data point that caused the alarm.
%dp	Description of the data point that caused the alarm. This is taken from the description (UCPTdescription) of the data point that caused the alarm in the Data Server.
%ua	The unit type of the data point that caused the alarm. This is taken from the <UCPTunit> property of the data point that caused the alarm, unless that data point is a SNVT_alarm or UNVT_alarm2. In those cases, this macro returns a blank string.
%nl	Linefeed. Enter this macro to insert a carriage return into your e-mail.

8.2.1.2.3 Active and Passive Alarm Condition Sets

Table 35 describes the properties that you must define within each <ActiveAlarm> and <PassiveAlarm> element. As described earlier in this chapter, each of these elements defines an active or passive alarm condition set for the Alarm Notifier.

If an input data point is updated and meets the conditions defined for any of the active condition sets, it will be considered an active alarm, and the active alarm destinations will be used for the alarm notification. If an input data point is updated and meets the conditions defined for any of the passive condition sets, it will be considered an passive alarm, and the passive alarm destinations will be used for the alarm notification.

The next section, *Active and Passive Alarm Destinations*, describes how you can define the active and passive destination sets for an Alarm Notifier.

Table 35 Active and Passive Alarm Conditions Set Properties

Property	Description
`<UCPTindex>`	The index number of the alarm condition set.

Property	Description
`<UCPTlevel>`	Enter an alarm level for the condition set, in the range 0-255. The level assigned to a condition set will determine which alarm destinations will be used when an alarm occurs that is based on that condition set. For each alarm destination you create, you will specify a range of levels. For example, you could set up one destination set for the alarm conditions using levels 0-125, and another for the alarm conditions using levels 126-255. Alarm condition sets assigned levels 0-125 would use the first destination, and alarm condition sets assigned level 126-255 would use the second destination set. **NOTE:** If you use the *i.*LON 100 Configuration Software to modify the configuration of an Alarm Notifier after creating it with the SOAP/XML interface, and the <UCPTlevel> property had been set to a value greater than 1, the <UCPTlevel> property will be reset to 0.
`<UCPTalarmText>`	The user-defined text will be used to describe the alarm condition in the Alarm Notifier's log file. This can be a maximum of 201 characters long.
`<UCPTactAlarmType>` for active conditon sets. `<UCPTpasAlarmType>` for passive conditon sets.	Specify one or more alarm types for this condition. If the status (UCPTpointStatus) of an input data point is updated and matches any of these types, then the alarm will be declared active or passive, depending on the conditon type. The valid alarm type identifiers are: AL_VALUE_INVALID, AL_CONSTANT, AL_OFFLINE, AL_NUL, AL_NO_CONDITION, AL_TOT_SVC_ALM_1. AL_TOT_SVC_ALM_2, AL_TOT_SVC_ALM_3, AL_LOW_LMT_CLR_1, AL_LOW_LMT_CLR_2, AL_HIGH_LMT_CLR_1, AL_HIGH_LMT_CLR_2, AL_LOW_LMT_ALM_1, AL_LOW_LMT_ALM_2, AL_HIGH_LMT_ALM_1, AL_HIGH_LMT_ALM_2, AL_FIR_ALM, AL_FIR_PRE_ALM, AL_FIR_TRBL, AL_FIR_SUPV, AL_FIR_TEST_ALM, AL_FIR_TEST_PRE_ALM, AL_FIR_ENVCOMP_MAX, AL_FIR_MONITOR_COND, AL_FIR_MAINT_ALERT You should consider using less severe conditions, such as AL_VALUE_INVALID or AL_OFFLINE, for your passive condition sets, and more severe conditions such as AL_HIGH_LMT_ALM_1 for your active conditon sets.

8.2.1.2.4 Active and Passive Alarm Destinations

You can define one or more <AlarmDest> elements per Alarm Notifier. These elements define the active and passive destinations for the Alarm Notifier.

You can optionally fill in the <UCPTdestEnable> property for each <AlarmDest> element. You can reference a SNVT_Switch data point by its <UCPTpointName> with this property. The <AlarmDest> will enabled if that data point is set to 100.0 1, or disabled if that data point is set to 0.0 0. You can set this data point with a LONWORKS switch or with the Event Scheduler application. In this fashion, you can enable or disable destination sets as you like.

Every <AlarmDest> should contain one or more <ActiveDest> and <PassiveDest> elements. Table 36 describes the properties you must define for each <ActiveDest> and <PassiveDest> element. Each <ActiveDest> element defines an active destination set for the Alarm Notifier. Each <PassiveDest> element defines a passive destination set for the alarm notifier.

The active destinations for an Alarm Notifier are used when the input data point is updated, and meets the conditions defined by any of the Alarm Notifier's active condition sets. The passive destinations for an Alarm Notifier are used when the input data point is updated, and meets any of the conditions defined by the Alarm Notifier's passive condition sets.

Table 36 Active and Passive Destination Properties

Property	Description
`<UCPTindex>`	The index number of the alarm destination set.
`<UCPTemailNickName>`	This optional property contains an e-mail nickname, as defined for an e-mail profile created for the Alarm Notifier. The e-mail profile to be used each time an alarm notification uses this destination set. **NOTE:** The *i.*LON Configuration Software will only allow you to display and modify one active, and one passive, alarm destination. The destinastions displayed will be the first listed in the XML files. If you use the *i.*LON 100 Configuration Software to modify the configuration of the Alarm Notifier, and these destinations are using different e-mail profiles, it will modify the Alarm Notfier to use the same e-mail profile for the first passive destination as it does for the first active destination.
`<UCPTpointName>`	The name of the output data point that will be updated when the active destination is used, and the e-mail for the alarm notification has been sent. If you want to create a destination that will automatically update the data point during an alarm notification without waiting for the e-mail to be sent, do not fill in the <UCPTe-mailNickName> property.
`<UCPTpointValue>`	The value, or value definition, that the output data point for the destination set will be updated to and the e-mail has been sent successfully by the Alarm Notifier. Value definitions are user-defined strings representing actual values. They can be added to a data point using the DataServerSet function.
`<UCPTminLevel>`	The minimum alarm level required for this destination to be used. The alarm level for an alarm notification is determined by the value assigned to the <UCPTlevel> property for of condition set that caused it.
`<UCPTmaxLevel>`	The maximum alarm level required for this destination to be used. The alarm level for an alarm notification is determined by the value assigned to the <UCPTlevel> property of the condition set that caused it.

Property	Description
`<UCPTnackDelay>`	The delay, in minutes, to wait for an alarm to be acknowledged before sending an e-mail to the e-mail profile for the destination. If the alarm is not acknowledged before this time expires, the e-mail profile will be used. The default value used if this property is not set is 0. In this case, the e-mail profile will be used as soon as the alarm occurs. The maximum is 65,535.

8.2.1.3 AlarmNotifierSet

Use the AlarmNotifierSet function to create new Alarm Notifiers, or to overwrite the configuration of exisiting Alarm Notifiers. The Alarm Notifiers to be created or written to are signified by a list of <Alarm> elements in the <Data> parameter. The properties that you must define within each <Alarm> element are the same, whether you are creating a new Alarm Notifier or modifying an existing Alarm Notifier. The previous section, *AlarmNotifierGet*, describes these properties.

NOTE: When modifying an existing Alarm Notifier, any optional properties left out of the input string will be erased. Old values will not be carried over, so you must fill in every property when writing to an Alarm Notifier, even if you are not changing all of the values.

You can create up to 40 Alarm Notifiers per *i.*LON 100. The first invocation of the AlarmNotifierSet function will generate the AlarmNotifier.XML file in the /root/config/software/Driver directory of your *i.*LON 100, if it does not already exist.

When creating or modifying an Alarm Notifier with AlarmNotifierSet, you may want to use output from AlarmNotifierGet as the basis for your <Data> parameter. You would then only need to modify the values of each property to match the new configuration you want, as opposed to re-creating an entire string like the one shown below, to generate your input.

The example below creates an Alarm Notifier that uses NVL_nviRequest as its input data point. This Alarm Notifier includes one e-mail profile that it can use each time an alarm notification occurs. It also has two ouput data points that can be updated when alarm notifications occur. Several factors determine which of the data points will be updated when the Alarm Notifier logs an alarm, including the status the input data point is updated to and the alarm level assigned to the alarm condition set.

<Data> Parameter

```
<Data>
  <iLONAlarmNotifier>
    <Alarm>
      <UCPTindex>9</UCPTindex>
      <UCPTdescription>Temperature Sensor Device</UCPTdescription>
      <UCPTfbName>Alarm Notifier- 9</UCPTfbName>
      <SCPTdelayTime>0.0</SCPTdelayTime>
      <UCPTsumLogSize>100</UCPTsumLogSize>
      <UCPThistLogSize>100</UCPThistLogSize>
      <UCPTlogFormat>LF_BINARY</UCPTlogFormat>
      <UCPTemailAggregTime>10</UCPTemailAggregTime>
      <Point>
        <UCPTindex>0</UCPTindex>
        <UCPTpointName>NVL_nviRequest</UCPTpointName>
        <UCPTalarmFlags>0 1 0 0 0 0 0</UCPTalarmFlags>
        <UCPTalarmGroup>1</UCPTalarmGroup>
        <UCPTpriority>7</UCPTpriority>
        <UCPTdescription></UCPTdescription>
      </Point>
      <Mail>
        <UCPTindex>0</UCPTindex>
        <UCPTemailNickName>Headquarters</UCPTemailNickName>
        <UCPTemailAddress>js@nova</UCPTemailAddress>
        <UCPTemailFormat>%al occured at %dy/%dm/%dd %pn </UCPTemailFormat>
        <UCPTemailSubject>Alarm Notifier 3: %ad</UCPTemailSubject>
        <UCPTemailAttachment>/root/Data/log1.csv</UCPTemailAttachment>
      </Mail>
      <ActiveAlarm>
        <UCPTindex>0</UCPTindex>
```

```
            <UCPTlevel>1</UCPTlevel>
            <UCPTalarmText>Log 30 percent full</UCPTalarmText>
            <UCPTactAlarmType>AL_ALM_CONDITION</UCPTactAlarmType>
          </ActiveAlarm>
          <PassiveAlarm>
            <UCPTindex>0</UCPTindex>
            <UCPTlevel>255</UCPTlevel>
            <UCPTalarmText>Normal Condition</UCPTalarmText>
            <UCPTpasAlarmType>AL_NUL</UCPTpasAlarmType>
          </PassiveAlarm>
          <AlarmDest>
           <UCPTindex>0</UCPTindex>
           <UCPTdestEnable>NVL_nviWeekday</UCPTdestEnable>
           <ActiveDest>
              <UCPTindex>0</UCPTindex>
              <UCPTemailNickName>Headquarters</UCPTemailNickName>
              <UCPTpointName>NVL_nvoDlClear</UCPTpointName>
              <UCPTpointValue>100.0 1</UCPTpointValue>
              <UCPTminLevel>2</UCPTminLevel>
              <UCPTmaxLevel>0</UCPTmaxLevel>
              <UCPTnackDelay>0</UCPTnackDelay>
           </ActiveDest>
           <PassiveDest>
              <UCPTindex>0</UCPTindex>
              <UCPTemailNickName>Headquarters</UCPTemailNickName>
              <UCPTpointName>NVL_nvoDlClear</UCPTpointName>
              <UCPTpointValue>0.0 0</UCPTpointValue>
           </PassiveDest>
          </AlarmDest>
        </Alarm>
      </iLONAlarmNotifier>
    <Data>
```

<Result> Parameter

```
<Result>
  <iLONAlarmNotifier>
    <Alarm>
      <UCPTindex>9</UCPTindex>
    </Alarm>
  </iLONAlarmNotifier>
</Result>
```

8.2.1.4 AlarmNotifierRead

Each time an Alarm Notifier causes an alarm notification, it will record an entry for that notification into its log file. You can use the AlarmNotifierRead function to retrieve some or all of the log entries that an Alarm Notifier has recorded. You must reference the Alarm Notifier to return log entries for by its index number in <Data> parameter you supply to the function.

You can specify which log entries the function will return by filling the properties described in Table 37 into the <Data> parameter you supply to the function. If you do not fill the parameters described in Table 37 into the <Data> parameter, all the entries in the log will be returned. **However, you should not attempt to return more than 100 log entries with a single call to this function.**

NOTE: You can find the log files in the /root/AlarmLog directory of the *i.*LON 100. 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.

Table 37 AlarmNotifierRead Input Properties

Parameter	Description
`<UCPTpointName>`	Enter the name of the data point you want to see log entries for. Leave this property blank to see log entries for all data points the Alarm Notifier is monitoring.
`<UCPTalarmLog>`	Enter HISTORICAL to return the contents of the Alarm History Log, which contains a log entry for every alarm notification made by the Alarm Notifier. Enter SUMMARY to return the contents of the Alarm Summary Log, which contains an entry for every active alarm notification made by the Alarm Notifier. It may only contain the most recent entry into the log, depending on how the <UCPTflags> property was defined for the Alarm Notifier when it was created.
`<UCPTcount>`	Use this field to specify the maximum number of log entries the function will return. If this property is not filled in, the function will return all log entries for the applicable data point, or data points, that occurred within the interval defined by the <UCPTstart> and <UCPTstop> properties. **NOTE:** You should not attempt to read more than 100 log entries with a single call to this function.

Parameter	Description
`<UCPTstart>` `<UCPTstop>`	Use these fields to specify a range for the log time of the entries that will be returned by the function. You can specify a start and stop time, or just a stop time. When reading the Alarm History Log, these fields indicate a range for the log time of the entries in the log, which is stored for each entry in the <UCPTlogTime> property. When reading the Alarm Summary Log, these fields indicate a range for the alarm time of the entries in the log, which is stored for each entry in the <UCPTalarmTime> property. If you specify a start and stop time and the number of log entries during this interval exceeds the maximum defined by the <UCPTcount> property, the function will return the first group of log entries recorded during the interval. If you only specify a start time, the function will return entries from the log starting at the start time until it reaches the end of the log file, or until it has returned the maximum number of entries (as defined by the <Count> property). If you only specify a stop time and the number of log entries during this interval exceeds the maximum defined by the <UCPTcount> property, the function will return the group of entries from the stop time going backwards in the log until the maximum number of log entries have been returned. If the <UCPTcount> property was not defined, the function will return all log entries in the log, going backward from the stop time. This may be useful for applications that need to read the newest information logged. If you do not enter a start or stop time, the function will return all log entries for the applicable data points, up to the maximum. You must enter the <UCPTstart> and <UCPTstop> properties as timestamps in local time, with appended time zone indicators to denote the difference between local time and UTC. For more information on this format, see *Local Times and Coordinated Universal Time* on page 6-12.The <UCPTstart> and <UCPTstop> properties must be entered as timestamps in local time, with an appended time zone indicator that denotes the difference between local time and UTC. For more information on this format, see *Local Times and Coordinated Universal Time* on page 6-12.

The following call to the AlarmNotifierRead function returns all log entries for data point NVL_AlarmGenIn1 the Alarm Notifier using index number 0 for alarms that were logged between 1-1-02 and 7-4-03. This call will return up to 200 log entries.

<Data> Parameter

```
<Data>
   <iLONAlarmNotifier>
      <AlarmLog>
         <UCPTindex>1</UCPTindex>
         <UCPTpointName> NVL_AlarmGenIn1</UCPTpointName>
         <UCPTalarmLog>HISTORICAL</UCPTalarmLog>
         <UCPTstart>2002-01-01T00:00:00.000+01:00</UCPTstart>
         <UCPTstop>2002-09-04T23:59:59.000+01:00</UCPTstop>
         <UCPTcount>5</UCPTcount>
      </AlarmLog>
   </iLONAlarmNotifier>
</Data>
```

<Result> Parameter

```
<Result>
   <iLONAlarmNotifier>
      <AlarmLog>
         <UCPTindex>0</UCPTindex>
         <UCPTpointName> NVL_AlarmGenIn1</UCPTpointName>
         <UCPTalarmLog>SUMMARY</UCPTalarmLog>
          <UCPTfileName>/root/AlarmLog/sumlog0.dat</UCPTfileName>
         <UCPTstart>2001-01-01T00:00:00.000+01:00</UCPTstart>
         <UCPTstop>2003-07-04T23:59:59.000+01:00</UCPTstop>
         <UCPTlastEvent>2001-2-31T23:59:59.566+01:00</UCPTlastEvent>
         <UCPTlogLevel>20.5</UCPTlogLevel>
         <UCPTtotalCount>154</UCPTtotalCount>
         <Element>
           <UCPTlogTime>2002-07-03T10:47:51.000+01:00</UCPTlogTime>
           <UCPTalarmTime>2002-05-05T1:12:15.000+01:00</UCPTalarmTime>
           <UCPTpointName>NVL_AlarmGenIn1</UCPTpointName>
           <UCPTlogSourceAddress>0.0</UCPTlogSourceAddress>
           <UCPTlocation>iLON</UCPTlocation>
           <UCPTalarmText>Normal Condition</UCPTalarmText>
           <UCPTpriority>1</UCPTpriority>
           <UCPTalarmGroup>0</UCPTalarmGroup>
           <UCPTalarmFlags>1 0 0 1 1 0 0</UCPTalarmFlags>
           <UCPTvalue>0.000000</UCPTvalue>
           <UCPTvalueDef></UCPTvalueDef>
           <UCPTunit></UCPTunit>
           <UCPTalarmType>PASSIVE</UCPTalarmType>
           <UCPTpointStatus>AL_NO_CONDITION</UCPTpointStatus>
           <UCPTalarmStatus>NACK</UCPTalarmStatus>
           <UCPTuserName>ilon</UCPTuserName>
           <UCPTdescription>Alarm Notifier Entry</UCPTdescription>
        </Element>
     </AlarmLog>
   </iLONAlarmNotifier>
</Result>
```

The <Result> parameter includes a series of properties for the Alarm Notifier referenced at the start of the <Data> parameter. These properties provide information about the Alarm Notifier and the log file the entries were read from. Table 38 describes these properties.

Table 38 AlarmNotifierRead Global Properties

Properties	Description
`<UCPTindex>`	The index number assigned to the Alarm Notifier.
`<UCPTalarmLog>`	The type of log requested (either HISTORICAL or SUMMARY).

Properties	Description
`<UCPTfileName>`	The name of the log file the Alarm Notifier is using.

`<UCPTstart>`
`<UCPTstop>`
	When reading the Alarm History Log, these properties contain timestamps indicating the log time of the first and last entries in the log file. When reading the Alarm Summary Log, these properties contain timestamps indicating the alarm time of the first and last entries in the log file. The order of the entries returned by the function will be sorted by log time for the Alarm History Log, and by alarm time for the Alarm Summary Log.
	These timestamps are displayed in local time, with appended time zone indicators that indicate the difference between local time and UTC. For more information on this, see *Local Times and Coordinated Universal Time* on page 6-12.

`<UCPTlastEvent>`	This property contains a timestamp indicating the last time an entry in the log file was deleted with the AlarmNotifierClear function, or the last time an entry in the log was modified with the AlarmNotifierWrite function. The timestamp is displayed in local time, with an appended time zone indicator that indicates the difference between local time and UTC. For more information on this, see *Local Times and Coordinated Universal Time* on page 6-12.
`<UCPTlogLevel>`	The log level of the Alarm Notifier's log file. This indicates the percentage of the log file's volume that has been filled. For example, the value 20.5 indicates that the log is 20.5% full.
`<UCPTtotalCount>`	This property contains the total number of entries contained in the alarm log read by the function.

The function also returns an <Element> element for each log entry that meets the criteria defined in the <Data> parameter you supplied to the function. Table 39 describes the properties that are listed within each <Element> element.

Table 39 AlarmNotifierRead Output Properties

Property	Description
`<UCPTlogTime>`	A timestamp indicating the time that the log entry was made. This timestamp is displayed in local time, with an appended time zone indicator indicating the difference between local time and UTC. For more information on this format, see *Local Times and Coordinated Universal Time* on page 6-12.
`<UCPTalarmTime>`	A timestamp indicating the time that the alarm occurred. This timestamp is displayed in local time, with an appended time zone indicator indicating the difference between local time and UTC.For more information on this format, see *Local Times and Coordinated Universal Time* on page 6-12.
`<UCPTpointName>`	The name of the data point that caused the alarm notification.

Property	Description
`<UCPTlogSourceAddress>`	The network address of the device the data point is local to. This is displayed using the following format: [Subnet.Node]
`<UCPTlocation>`	The location of the data point.
`<UCPTalarmText>`	The alarm text for the alarm. This text can be specified for an Alarm Notifier using the AlarmNotifierSet function.
`<UCPTpriority>`	The priority level assigned to the data point that caused the alarm. The priority level is an integer between 0 (high priority) and 255 (low priority). You can use this priority level to sort the alarms with the summary log view, or with the *i.*LON 100 Web page.
`<UCPTalarmGroup>`	The alarm group of the alarm. This may be useful when sorting alarms.
`<UCPTalarmFlags>`	The alarm flags string defined for the data point that caused the alarm. For more information on this property, see *AlarmNotifierGet* on page 8-6.
`<UCPTvalue>`	The value of the data point that caused the alarm notification.
`<UCPTvalueDef>`	The value definition being used by the data point. Values definitions are strings representing preset values. They can be established when a data point is added to the Data Server. If this property does not appear, then the data point is not currently using a value definition. For more information on value definitions, see Chapter 5, *Data Server*.
`<UCPTunit>`	The unit type defined for the data point that caused the alarm.
`<UCPTalarmType>`	ACTIVE or PASSIVE. This indicates whether the alarm was an active or passive alarm. The conditions that determine whether an alarm is active or passive are defined when the Alarm Notifier is created. For more information, see *Active and Passive Alarm Condition Sets* on page 8-15.
`<UCPTpointStatus>`	The status of the data point that caused the alarm notification. For information on how you can determine which point statuses cause alarm notifications, see *Active and Passive Alarm Conditions Set Properties* on page 8-15.

Property	Description
`<UCPTalarmStatus>`	The status of the alarm. This can be AUTO_ACK or MANUAL_ACK for an acknowlegded alarm that has not been removed from the active alarms list, AUTO_CLEAR or MANUAL_CLEAR for an alarm that has been acknowledged but not removed from the active alarm list, and NACK for an alarm that has not yet been acknowledged. You can clear or acknowledge alarms manually with the *AlarmNotifierWrite* function. For more information, see *AlarmNotifierWrite* on page 8-26. Alarms may be cleared or acknowledged automatically depending on how the <UCPTflags> property was defined for the Alarm Notifier when it was created.
`<UCPTuserName>`	The name of the user who acknowledged the alarm. Alarms can be acknowledged with the AlarmNotifierWrite function.
`<UCPTdescription>`	The comment entered into the log entry for the log. You can enter comments into the log with the AlarmNotifierWrite function.

8.2.1.5 AlarmNotifierWrite

You can use the AlarmNotifierWrite function to acknowledge, or comment on, an log entry for an Alarm Notifier. Table 40 describes the input properties you can define in the <Data> parameter to acknowledge an alarm.

Table 40 AlarmNotifierWrite Input Properties

Property	Description
`<UCPTindex>`	The index number of the Alarm Notifier that generated the alarm notification.
`<UCPTpointName>`	The name of the data point that caused the alarm notification.
`<UCPTalarmTime>`	A timestamp indicating the time that the alarm occurred. You must enter this timestamp in local time, with an appended time zone indicator that shows the difference between local time and UTC. For more information on this format, see *Local Times and Coordinated Universal Time* on page 6-12.
`<UCPTuserName>`	The user name of the person acknowledging the alarm. This will be logged in the log file. This can be a maximum of 31 characters long.
`<UCPTdescription>`	Enter a comment to be recorded in the log file entry for this alarm. This can be a maximum of 228 characters long.

Property	Description
`<UCPTalarmStatus>`	You can select one of four parameters to change the alarm status entered in the log: • **MANUAL_CLEAR :** Alarm will be acknowledged and removed from the active list. • **MANUAL_ACK:** Alarm will be acknowledged, but not removed from the active list. • **NACK:** Alarm will not be acknowledged or removed from the active list. However, the comment entered for the <UCPTcomment> property will be entered into the log. • **AUTO_ACK:** If the status of an alarm reads AUTO_ACK, it indicates that the alarm was automatically acknowledged by the Alarm Notifier when it occurred. You can cause an Alarm Notifier to automatically acknowledge all alarms for a data point by setting Byte 5 of the <UCPTalarmFlags> property for the data point to 1 when you create your Alarm Notifier with AlarmNotifierSet. You can still enter comments for the log file using this function if an alarm was automatically acknowledged. For more information on the <UCPTalarmFlags> property, see *Input Data Points* on page 8-10.

The following example acknowledges an alarm caused by the NVL_AlarmGenIn1 data point.

<Data> Parameter

```
<Data>
  <iLONAlarmNotifier>
    <AlarmLog>
    <UCPTindex>0</UCPTindex>
    <Element>
       <UCPTpointName>NVL_AlarmGenIn1</UCPTpointName>
       <UCPTalarmTime>2002-01-01 Z1:05:03.000+01:00</UCPTalarmTime>
       <UCPTuserName>Ben Ross</UCPTuserName>
       <UCPTdescription>I'll fix it this evening</UCPTdescription>
       <UCPTalarmStatus>MANUAL_ACK</UCPTalarmStatus>
       </Element>
     </AlarmLog>
  </iLONAlarmNotifier>
</Data>
```

<Result> Parameter

```
<Result>
 <iLONAlarmNotifier>
   <AlarmLog>
      <UCPTindex>0</UCPTindex>
      <UCPTalarmLog>SUMMARY</UCPTalarmLog>
      <UCPTfileName>/root/AlarmLog/sumlog0.csv</UCPTfileName>
      <UCPTstart>2001-01-01Z1:05:03.000+01:00</UCPTstart>
      <UCPTstop>2003-01-01Z1:05:03.000+01:00</UCPTstop>
      <UCPTlogLevel>0.5</UCPTlogLevel>
   </AlarmLog>
 </iLONAlarmNotifier>
</Result>
```

Table 41 describes the properties returned by the AlarmNotifierWrite function in the <Result> parameter.

Table 41 AlarmNotifierWrite Output Properties

Property	Description
`<UCPTindex>`	The index number of the Alarm Notifier affected by the function.
`<UCPTalarmLog>`	The type of log file affected by the function: SUMMARY or HISTORICAL.
`<UCPTfileName>`	The name and path of the log file affected by the function.

`<UCPTstart>`
`<UCPTstop>`
	The <UCPTstart> and <UCPTstop> properties indicate the time the alarm time of the first and last entries in the log file. These timestamps are shown in local time, with appended time zone indicators that indicate the difference between local time and UTC. For more information on this format, see *Local Times and Coordinated Universal Time* on page 6-12.

`<UCPTlogLevel>`	The volume of the log file currently being used. For example, the value 20.5 indicates that the log file is 20.5% full.

8.2.1.6 AlarmNotifierClear

Use the AlarmNotifierClear function to clear a group of log entries from an Alarm Notifier log file. This function only deletes the log entries. You can delete the Alarm Notifier itself with the AlarmNotifierDelete function.

You can specify which alarm entries are to be cleared out by filling the properties described in Table 42 into the <Data> parameter you supply to the function. If you do not fill in these properties, the entire alarm log will be cleared.

Table 42 AlarmNotifierClear Input Properties

Parameter	Description
`<UCPTindex>`	The index number of the Alarm Notifier to be affected.
`<UCPTpointName>`	The name of the data point whose log entries are to be deleted. If no data point name is specified, log entries for all data points will be deleted.
`<UCPTcount>`	Use this field to specify the maximum number of log entries the function will delete. If you do not fill in this property, all log entries for the applicable data point (or data points) will be cleared.
`<UCPTstart>` `<UCPTstop>`	Use these fields to specify a time range for the alarm time of each log entry to be deleted. You can specify a start and stop time, or just a stop time. If you specify a start and stop time and the number of log entries during this interval exceeds the count entered, the function will clear out the first group of log entries recorded during that interval. If you only specify a stop time and the number of log entries before that time exceeds the count entered, the function will clear out the first group of log entries that recorded during that interval. If you do not enter a start or stop time, the function will clear out all log entries for the applicable data points, up to the maximum. You must enter these properties as timestamps in local time, with appended time zone indicators that denote the difference between local time and UTC. For more information on this format, see *Local Times and Coordinated Universal Time* on page 6-12.

The following call to the AlarmNotifierClear function deletes all log entries for data point NVL_nviBgtVa from the Alarm Notifier with index number 0 that occurred between 1/31/2001 at 14:30 and 2/28/2001 at 14:30. Since the count entered is 200, it will delete the first 200 log entries if the total log entries for the time span selected exceeds 200.

<Data> Parameter

```
<Data>
   <iLONAlarmNotifier>
      <AlarmLog>
         <UCPTindex>0</UCPTindex>
         <UCPTpointName>NVL_nviBgtVa</UCPTpointName>
         <UCPTstart>2001-01-31T14:30:00.000+03:00</UCPTstart>
         <UCPTstop>2001-02-28T14:30:00.000+03:00</UCPTstop>
         <UCPTcount>200</UCPTcount>
      </AlarmLog>
   </iLONAlarmNotifier>
</Data>
```

<Result> Parameter

```
<Result>
   <iLONAlarmNotifier>
      <AlarmLog>
         <UCPTindex>0</UCPTindex>
         <UCPTalarmLog>SUMMARY</UCPTalarmLog>
         <UCPTfileName>/root/AlarmLog/sumlog0.dat</UCPTfileName>
         <UCPTstart>2001-01-31T14:30:00.000+03:00<UCPTstart>
         <UCPTstop>2001-02-31T14:29:59.000+03:00<UCPTstop>
         <UCPTlogLevel>20.5</UCPTlogLevel>
      </AlarmLog>
   </iLONAlarmNotifier>
</Result>
```

Table 43 describes the properties returned by the AlarmNotifierClear function in the <Result> parameter.

Table 43 AlarmNotifierClear Output Properties

Property	Description
`<UCPTindex>`	The index number of the Alarm Notifier affected by the function.
`<UCPTalarmLog>`	The type of log file affected by the function: SUMMARY or HISTORICAL.
`<UCPTfileName>`	The name and path of the log file affected by the function.

`<UCPTstart>`
 	
`<UCPTstop>`
	The <UCPTstart> and <UCPTstop> properties indicate the alarm times of the first and last log entries deleted by the function. These properties are displayed as timestamps in local time, with appended time zone indicators that indicate the difference between local time and UTC. For more information on this format, see *Local Times and Coordinated Universal Time* on page 6-12.

`<UCPTlogLevel>`	The volume of the log file currently being used. For example, the value 20.5 indicates that the log file is 20.5% full.

8.2.1.7 AlarmNotifierDelete

You can use the AlarmNotifierDelete function to delete an Alarm Notifier. You must reference the Alarm Notifier to be deleted by its index number in the <Data> parameter you supply to the function, as in the example below.

<Data> Parameter

```
<Data>
  <iLONAlarmNotifier>
    <Alarm>
      <UCPTindex>9</UCPTindex>
    </Alarm>
  </iLONAlarmNotifier>
</Data>
```

<Result> Parameter

```
<Result>
  <iLONAlarmNotifier>
    <Alarm>
      <UCPTindex>9</UCPTindex>
    </Alarm>
  </iLONAlarmNotifier>
</Result>
```

9 Analog Function Block

You can use Analog Function Blocks to perform a variety of statistical operations on the values of the data points in your network, and store the result of each operation in an output data point. You can perform these operations on as many input data points as you like per Analog Function Block. The operations you can perform on them include determining the average value of the input data points, the maximum value of the input data points, the minimum value of the input data points, the sum of the input data point values, and several others. Each operation is described in detail later in this chapter.

You can also select a comparison function as your operation. In this case, the Analog Function Block will compare the value of all the input data points to the value of a data point selected as the compare data point. You can choose from a variety of comparisons that an Analog Function Block can perform between the data points, including *Greater Than*, *Less Than*, and *Equal To*. The Analog Function Block will compare the values of the compare and input data point using that comparison, and update the output data point to a True or False value based on the result of that comparison.

If you are using a comparison function, and your Analog Function Block has multiple input data points, you can specify a percentage. If that percentage of the comparisons between the input and compare data points returns True, the output data point will be set to True. Otherwise, it will be set to False.

For example, consider a case where an Analog Function Block has five input data points and is using *Greater Than* as the comparison function. Assume that the percentage is set to 50%. If the value of the 50% (at least three) of the input data points is greater than the value of the compare data point, the output data point will be set to True. Otherwise, it will be set to False.

The Analog Function Block will perform the operation you have selected for it each time any of its input data points are updated, or at a timed interval you specify. You could use these calculated values as a part of a control system or to monitor alarm conditions based on multiple inputs.

NOTE: Analog Function Blocks are not supported by, and can not be configured with, the *i.*LON 100 Configuration Software.

9.1 AnalogFB.XML

The AnalogFB.XML file stores the configuration of the Analog Function Blocks that you have added to the *i.*LON 100. You can create up to 20 Analog Function Blocks on your *i.*LON 100.

Each Analog Function Block is signified by an <AnalogFB> element in the XML file. You can create Analog Function Blocks with the AnalogFBSet function, or by manually editing the AnalogFB.XML file and downloading it to the *i.*LON 100 via FTP. The sections following this example provide instructions and guidelines to assist you when doing so.

The following represents a sample AnalogFB.XML file for an *i.*LON 100 with one defined Analog Function Block. This Analog Function Block determines the maximum value between two data points, NVL_nviClaValue_1 and NVL_nviClaValue_2**,** and stores that value in the value field of a data point called NVL_nvoClsValue_1.

```
<?xml version="1.0" ?>
  <iLONAnalogFB>
    <SCPTobjMajVer>1</SCPTobjMajVer>
```

```
   <SCPTobjMinVer>1</SCPTobjMinVer>
   <AnalogFB>
      <UCPTindex>0</UCPTindex>
      <UCPTlastUpdate>2002-06-02T09:16:36Z</UCPTlastUpdate>
      <UCPTdescription>Bielefeld</UCPTdescription>
      <UCPTfbName>Analog Fn Block- 0</UCPTfbName>
      <UCPTcompFunction>FN_GT</UCPTcompFunction>
      <UCPTmajorityValue>100</UCPTmajorityValue>
      <UCPTtrueThreshold />
      <UCPToutputFunction>FN_MAX</UCPToutputFunction>
      <SCPTminRnge>10.0</SCPTminRnge>
      <SCPTmaxRnge>80.0</SCPTmaxRnge>
      <UCPTcalculationInterval>0.0</UCPTcalculationInterval>
      <SCPTovrBehave>OV_DEFAULT</SCPTovrBehave>
      <SCPTovrValue>0</SCPTovrValue>
      <UCPTpollOnResetDelay>0.0</UCPTpollOnResetDelay>
      <InputDataPoint>
         <Point>
            <UCPTindex>0</UCPTindex>
            <UCPTpointName>NVL_nviClaValue_1</UCPTpointName>
            <UCPTfieldName>value</UCPTfieldName>
            <UCPTpollRate>0</UCPTpollRate>
         </Point>
         <Point>
            <UCPTindex>1</UCPTindex>
            <UCPTpointName>NVL_nviClaValue_2</UCPTpointName>
            <UCPTfieldName>value</UCPTfieldName>
            <UCPTpollRate>0</UCPTpollRate>
          </Point>
      </InputDataPoint>
      <CompareDataPoint>
         <UCPTpointName>NVL_nvoClsValue_2</UCPTpointName>
         <UCPTfieldName>value</UCPTfieldName>
         <UCPTpollRate>0</UCPTpollRate>
      </CompareDataPoint>
      <OutputDataPoint>
         <UCPTpointName>NVL_nvoClsValue_1</UCPTpointName>
         <UCPTfieldName>value</UCPTfieldName>
      </OutputDataPoint>
   </AnalogFB>
</iLONAnalogFB>
```

9.2 Creating and Modifying the AnalogFB.XML File

You can create and modify the AnalogFB.XML configuration file with the AnalogFBSet SOAP function. The following section, *Analog Function Block SOAP Interface*, describes how to use the AnalogFBSet function and the other SOAP functions provided for the Analog Function Block application.

Alternatively, you can create and modify the AnalogFB.XML file manually with an XML editor and download it to the *i.*LON 100 via FTP. Echelon does not recommend this, as the *i.*LON 100 will require a reboot to read the configuration of the downloaded file. Additionally, the *i.*LON 100 performs error checking on all SOAP messages it receives before writing to the XML file. It will not perform error checking on any XML files you download via FTP, and thus the application may not boot properly.

However, if you plan to create the XML file manually, you should review the rest of this chapter first, as it describes the elements and properties in the XML file that define each Alarm Notifier's configuration. For instructions on creating or modifying an XML file manually, see *Manually Modifying an XML Configuration File* on page 15-1.

9.2.1 Analog Function Block SOAP Interface

The SOAP interface for the Analog Function Block application includes four functions. Table 44 lists and describes these functions. For more information on each function, see the sections following Table 44.

Table 44 Analog Function Block

Function	Description
AnalogFBList	Use this function to generate a list of the Analog Function Blocks that you have added to the *i.*LON 100. For more information, see *AnalogFBList* on page 9-4.
AnalogFBGet	Use this function to return the configuration of an Analog Function Block. For more information, see *AnalogFBGet* on page 9-5.
AnalogFBSet	Use this function to create an Analog Function Block, or to overwrite the configuration of an existing Analog Function Block. For more information, see *AnalogFBSet* on page 9-14.
AnalogFBDelete	Use this function to delete an Analog Function Block. For more information, see *AnalogFBDelete* on page 9-15.

9.2.1.1 AnalogFBList

Use the AnalogFBList function to retrieve a list of the Analog Function Blocks that you have added to the *i.*LON 100. The AnalogFBList function takes an empty string as its <Data> parameter, as in the example below.

The function returns the major and minor version numbers of the firmware implementation the Analog Function Block application is using in the <Result> parameter. The <Result> parameter also includes an <AnalogFB> element for each Analog Function Block that you have added to the *i.*LON 100. The next section, *AnalogFBGet*, describes the properties included in each of these elements.

You could use the list of <AnalogFB> elements returned by this function as input for the AnalogFBGet function. The AnalogFBGet function would then return the configuration of each Analog Function Block included in the list.

<Data> Parameter

```
Empty String
```

<Result> Parameter

```
<Result>
    <iLONAnalogFB>
       <SCPTobjMajVer>1</SCPTobjMajVer>
       <SCPTobjMinVer>1</SCPTobjMinVer>
       <AnalogFB>
         <UCPTindex>0</UCPTindex>
         <UCPTlastUpdate>2002-06-02T19:16:36Z</UCPTlastUpdate>
         <UCPTdescription>Maximum Temperature</UCPTdescription>
         <UCPTfbName>Analog Fn Block- 0</UCPTfbName>
       </AnalogFB>
       <AnalogFB>
         <UCPTindex>1</UCPTindex>
         <UCPTlastUpdate>2002-06-26T10:10:55Z</UCPTlastUpdate>
         <UCPTdescription>Average Temperature</UCPTdescription>
         <UCPTfbName>Analog Fn Block - 1</UCPTfbName>
       </AnalogFB>
    </iLONAnalogFB>
</Result>
```

9.2.1.2 AnalogFBGet

You can use the AnalogFBGet function to retrieve the configuration of any Analog Function Block that you have added to the *i.*LON 100. You must reference the Analog Function Block whose configuration is to be displayed by its index number in the <Data> parameter you supply to the function, as in the example below.

<Data> Parameter

```
<Data>
  <iLONAnalogFB>
    <AnalogFB>
        <UCPTindex>1</UCPTindex>
    </AnalogFB>
  </iLONAnalogFB>
</Data>
```

<Result> Parameter

```
<Result>
   <iLONAnalogFB>
       <AnalogFB>
         <UCPTindex>1</UCPTindex>
          <UCPTlastUpdate>2002-06-02T09:06:36Z</UCPTlastUpdate>
         <UCPTdescription>Maximum Temperature</UCPTdescription>
         <UCPTfbName>Analog Fn Block- 1</UCPTfbName>
         <UCPTcompFunction>FN_GT</UCPTcompFunction>
         <UCPTmajorityValue>100</UCPTmajorityValue>
         <UCPTtrueThreshold></UCPTtrueThreshold>
         <UCPToutputFunction>FN_MAX</UCPToutputFunction>
         <SCPTminRnge>10.0</SCPTminRnge>
         <SCPTmaxRnge>80.0</SCPTmaxRnge>
         <UCPTcalculationInterval>0.0</UCPTcalculationInterval>
         <SCPTovrBehave>OV_DEFAULT</SCPTovrBehave>
         <SCPTovrValue>0</SCPTovrValue>
         <UCPTpollOnResetDelay>0.0</UCPTpollOnResetDelay>
         <InputDataPoint>
              <Point>
                <UCPTindex>0</UCPTindex>
                <UCPTpointName>NVL_nviClaValue_1</UCPTpointName>
                <UCPTfieldName>value</UCPTfieldName>
                <UCPTpollRate>0</UCPTpollRate>
              </Point>
              <Point>
                <UCPTindex>1</UCPTindex>
                <UCPTpointName>NVL_nviClaValue_2</UCPTpointName>
                <UCPTfieldName>value</UCPTfieldName>
                <UCPTpollRate>0</UCPTpollRate>
              </Point>
         </InputDataPoint>
         <CompareDataPoint>
              <UCPTpointName>NVL_nvoClsValue_2</UCPTpointName>
              <UCPTfieldName>value</UCPTfieldName>
              <UCPTpollRate>0</UCPTpollRate>
         </CompareDataPoint>
         <OutputDataPoint>
              <UCPTpointName>NVL_nvoClsValue_1</UCPTpointName>
              <UCPTfieldName>value</UCPTfieldName>
         </OutputDataPoint>
      </AnalogFB>
   </iLONAnalogFB>
</Result>
```

The function returns an <AnalogFB> element for each Analog Function Block referenced in the <Data> parameter you supplied to the function. The properties included in each <AnalogFB> element are initially defined when the Analog Function Block is created. You can write to them with the AnalogFBSet function. Table 45 describes these properties.

For more information on the AnalogFBSet function, see *AnalogFBSet* on page 9-14.

Table 45 AnalogFBGet Output Properties

Property	Description
`<UCPTindex>`	The index number assigned to the Analog Function Block must be in the range 0-32,767. As mentioned earlier, you can use the AnalogFBSet function to create a new Analog Function Block, or to modify an existing Analog Function Block. If you do not specify an index number in the <Data> parameter you supply to AnalogFBSet, the function will create a new Analog Function Block using the first available index number. If you specify an index number that is already being used, the function will overwrite the configuration of the Analog Function Block using that index number with the settings defined in the <Data> parameter.
`<UCPTlastUpdate>`	A timestamp indicating the last time the configuration of the Analog Function Block was updated. This timestamp uses the following format: YYYY-MM-DD**T**HH:MM:SS**Z** The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Analog Function Block was last updated. The second segment (**T**HH:MM:SS) represents the time of day the configuration of the Analog Function Block was last updated, in UTC (Coordinated Universal Time). UTC is the current term for what was commonly referred to as Greenwich Meridian Time (GMT). Zero (0) hours UTC is midnight in Greenwich England, which lies on the zero longitudinal meridian. Universal time is based on a 24 hour clock, therefore, an afternoon hour such as 4 pm UTC would expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002.
`<UCPTdescription>`	A description of the Analog Function Block. This can be a maximum of 228 characters long.
`<UCPTfbName>`	The functional block name assigned to the Analog Function Block in LONMAKER. You can write to this property, but each time you use the *i.*LON 100 Configuration Software to view the Analog Function Block, it will be reset to match the functional block name defined in LONMAKER.

Property	Description
`<UCPTcompFunction>`	This property defines the comparison function the Analog Function Block will use to compare the values of the compare data point and the input data points. This function will only be used if the <UCPToutputFunction> selected is FN_COMPARE, FN_OR or FN_AND, and if the <UCPTtrueThreshold> property is not defined. These properties are described later in the table. When this function is used, the output data point will be updated to a True or False value depending on the results of the comparisons made with this function. If more than one input data point is defined for the Analog Function Block, you can specify a percentage with the <UCPTmajorityValue> property. If that percentage of the input data points return True, the output data point will be updated to True. Otherwise, it will be updated to False. The <UCPTmajorityValue> property is described in more detail later in this table. For descriptions of the comparison functions you can use with your Analog Function Block, see *Comparison Functions* on page 9-10.
`<UCPTmajorityValue>`	The percentage of input data points whose comparison result with the compare data point (or with the value of the <UCPTtrueThreshold> property, if it is defined) must be True in order for the output data point is set to True. The comparison to be performed between the input and compare data point values is determined by the <UCPTcompFunction> selected. For example, if this field is set to 30.0, 30% of the input data points must return True in order for the output data point to be set to True. This field has a range of 0.0 to 100.0.
`<UCPTtrueThreshold>`	This property specifies the compare value to be used with the input data point when the comparison function selected for the Analog Function Block is FN_OR, FN_AND or FN_COMPARE. This property will only be used if the input data point(s) uses a scalar or enumeration value. This property can not be used if any of the input data point use the format type SNVT_switch. If this property is not defined, all the comparisons will made with the value of the compare data point. You can select a compare data point by filling in the <CompareDataPoint> element, which is described later in the table. Scenarios that may assist you in understanding how to use this property follow the *Comparison Functions* section on page 9-10.
`<UCPToutputFunction>`	The output function for the Analog Function Block. This determines the operation the Analog Function Block will perform each time its data points are updated, and how the value of the Analog Function Block's output data point will be determined. For descriptions of the output functions you can use with your Analog Function Block, see *Output Functions* on page 9-10.
`<SCPTminRnge>`	The minimum value that the output data point can be assigned. This value is initially taken from the SCPT resource files.
`<SCPTmaxRnge>`	The maximum value that the output data point can be assigned. This value is initially taken from the SCPT resource files.

Property	Description
`<UCPTcalculationInterval>`	The delay, in seconds (0.0 to 6553.0), that must elapse between updates to the Analog Function Block's output data point. This may be useful if you have multiple input data points, as setting a long interval here could cause the Analog Function Block to only update the output data point when all inputs have been received. If you use the default value of 0.0, the the Analog Function Block will update the output data point each time any of the input data points are updated.
`<SCPTovrBehave>`	A value to define the behavior of the output data point when an override request is received for the Analog Function Block. The valid range for this property is any value within the defined limits of SNVT_override. Enter OV_SPECIFIED to assign the output data point an override value when this occurs. You can specify the value to be used by filling in the <SCPTovrValue> property. If you do not fill in this property, the application will maintain its last setting when an override occurs.
`<SCPTovrValue>`	The value the output data point will be assigned when it is overridden, and the <SCPTovrBehave> property is set to OV_SPECIFIED.
`<UCPTpollOnResetDelay>`	The delay, in seconds, the Analog Function Block wait after a reset before polling the values of the input data points. When this value is 0, the Analog Function Block will resume polling the input data points at the rate specified by the <UCPTpollRate> property after a reset. This field has a range of 0.0-6553.0.

Property	Description
`<Point>`	You can specify as many input data points as you want per Analog Function Block. The input data points for an Analog Function Block are defined by a list of <Point> elements in the <Data> parameter supplied to the function. For each element, you must specify an index number to be used within the Analog Function Block (UCPTindex), the name of the data point (UCPTpointName), and the interval to use when polling the data point's value (UCPTpollRate). The poll rate must be specified as an integer between 0-6553. If the input data point is a structure, you must also specify the name of the field to use when performing comparisons with the data point (UCPTfieldName). The value of the selected field for each input data point will be used to generate a value for the output data point. This value assigned to the output data point will vary, depending on the output function (UCPToutputFunction) selected for the Analog Function Block. **NOTE:** You should note that other *i.*LON 100 applications may cause the Data Server to poll this data point's value as well. The poll rate specified by these applications should be compatible with each other. For example, if an Analog Function Block is polling a data point every 15 seconds, and the Data Logger is polling that data point every 10 seconds, then the Data Server will have to poll the value of the data point every five seconds to ensure that each application gets a current value for each poll. It is important to note this as you set poll rates for various applications, as you may end up causing more polls than is efficient on your network. For example, if an Analog Function Block is polling a data point every 9 seconds and a Data Logger is polling a data point every 10 seconds, the Data Server would have to poll the data point every second to ensure that each application polls for a current value. This may create a significant amount of undesired traffic.
`<CompareDataPoint>`	This element defines the compare data point this Analog Function Block will use. You must specify the name of the data point (UCPTpointName), the name of the field to use when making comparisons with the data point (UCPTfieldName) if it is a structure, and the interval to use when polling the data point's value (UCPTpollRate). The value of this data point will be compared to the value of each input data point when the output function selected for the Analog Function Block is FN_COMPARE, FN_AND or FN_OR. The comparison to perform is determined by the <UCPTcompFunction> property, and the result of this comparison will be stored in the output data point. This value will not be used in comparisons if the <UCPTtrueThreshold> property is defined.

Property	Description
`<OutputDataPoint>`	This element defines the output data point for this function block. You must specify the <UCPTpointName> assigned to the output data point within this element. The value of this data point will be updated with the result of each comparison or statistical operation that the Analog Function Block performs.

9.2.1.2.1 Output Functions

Table 46 lists and describes the output functions you can use to fill in the <UCPToutputFunction> property. You must reference each function by the identifier listed in the table.

The function selected here determines the value that the Analog Function Block will assign to the output data point.

Table 46 Output Function Identifiers

Identifier	Value Assigned To The Output Data Point
`FN_MAX`	Maximum value of all input data points.
`FN_MIN`	Minimum value of all input data points.
`FN_SUM`	The sum of the values of all input data points.
`FN_AVERAGE`	The average of the values of the input data points.
`FN_COMPARE`	The result of the last comparison between the input data point(s) and the compare data point (or the value assigned to the <UCPTtrueThreshold> property, if it is defined). If this is selected, you must also select a comparison function by filling in the <UCPTcompFunction> property. For an example of how you could use this function, see *FN_COMPARE Example* on page 9-12.
`FN_AND`	This function reports True when all the input data points are True. The definition of a True input depends on the data point type. If the input type is SNVT_switch, the input is True if the value and state fields are non-zero. If the input type is a structure other than SNVT_switch, the Boolean threshold is undefined, and FN_AND should not be used. If the input data point(s) type is a scalar or enumeration value, the function reports True if all the comparisons made by the comparison function for the analog function block are True. For an example of how you could use the FN_AND output function in this way, see *FN_AND Example* on page 9-11.
`FN_OR`	This function reports True when any of the input data points are True. The definition of a True input depends on the data point type. If the input type is SNVT_switch, the input is True if the state and value fields are non-zero. If the input type is a structure other than SNVT_switch, the Boolean threshold is undefined, and FN_OR should not be used. If the input data point(s) type is a scalar or enumeration value, the function reports True if any of the comparisons made by the comparison function for the analog function block are True. For an example of how you could use the FN_OR function in this way, see *FN_OR Example* on page 9-12.

Identifier	Value Assigned To The Output Data Point
`FN_NUL`	Value not available.

9.2.1.2.2 Comparison Functions

Table 47 lists and describes the comparison functions you can use to fill in the <UCPTcompFunction> property. You must reference each function by the identifier listed in the table.

Table 47 Comparison Function Identifiers

Identifier	Description
`FN_GT`	Greater than. Returns True if the value of the input data point is greater than that of the compare data point (or the value assigned to the <UCPTtrueThreshold> property, if it is defined).
`FN_LT`	Less than. Returns True if the value of the input data point is less than that of the compare data point (or the value assigned to the <UCPTtrueThreshold> property, if it is defined).
`FN_GE`	Greater than or equal to. Returns True if the value of the input data point is greater than or equal to that of the compare data point (or the value assigned to the <UCPTtrueThreshold> property, if it is defined).
`FN_LE`	Less than or equal to. Returns True if the value of the input data point is less than or equal to that of the compare data point (or the value assigned to the <UCPTtrueThreshold> property, if it is defined).
`FN_EQ`	Equal. Returns True if the value of the input data point is equal to that of the compare data point (or the value assigned to the <UCPTtrueThreshold> property, if it is defined).
`FN_NE`	Not equal. Returns True if the value of the input data point is not equal to that of the compare data point (or the value assigned to the <UCPTtrueThreshold> property, if it is defined).
`FN_NUL`	Value not available. Returns True if the value of the input data point is not available.

9.2.1.2.3 FN_AND Example

<UCPToutputFunction>:	FN_AND
<UCPTcompFunction>:	FN_GT

In this example, there are four input data points and one compare data point, all of the type SNVT_count. There is one output data point, of the type SNVT_Switch.

Because the output function is FN_AND, the comparisons made with all the input data points must return True in order for the output data point to be set to True. The comparison function is FN_GT, so the value of each input data point must be greater than the value of the compare data point, or the <UCPTtrueThreshold> value if it is defined, for this to happen. If the <UCPTtrueThreshold> property is defined, then the value of the compare data point is not used in the comparison.

Table 48 lists several case scenarios that show when these functions might would evaluate to True.

Table 48 FN_AND Examples

Input 1	Input 2	Input 3	Input 4	Value of Compare Data Point	UCPTtrueThreshold	Output
9	11	12	13	Does not matter since <UCPTtrueThreshold> is defined.	10	0.0 0
20	30	40	50	Does not matter since <UCPTtrueThreshold> is defined.	10	100.0 1
20	30	40	50	35	EMPTY	0.0 0
70	80	40	50	35	EMPTY	100.0 1

9.2.1.2.4 FN_OR Example

<UCPToutputFunction>:	FN_OR
<UCPTcompFunction>:	FN_LT

In this example, there are four input data points and one compare data point, all of the type SNVT_count. There is one output data point, of the type SNVT_switch.

Because the output function is FN_OR, and the comparison function is FN_LT, one of the values of the data inputs must be less than the value of the compare data point, or the <UCPTtrueThreshold> value if it is defined, for the output data point to be set to True. If the <UCPTtrueThreshold> property is defined, then value of the compare data point is not used in the comparison.

Table 49 lists several case scenarios that show when these two functions might evaluate to True.

Table 49 FN_OR Examples

Input 1	Input 2	Input 3	Input 4	Value of Compare Data Point	UCPTtrueThreshold	Output
9	11	12	13	Does not matter since <UCPTtrueThreshold> is defined.	10	100.0 1
20	30	40	50	35	EMPTY	0.0 0
20	30	40	50	35	EMPTY	100.0 1
20	30	40	50	35	EMPTY	100.0 1

9.2.1.2.5 FN_COMPARE Example

<UCPToutputFunction>:	FN_COMPARE
<UCPTcompFunction>:	FN_EQ
<UCPTmajorityValue>:	100

In this example, there are four input data points and one compare data point, all of the type SNVT_count. There is one output data point, of the type SNVT_switch.

Because the <UCPTmajorityValue> is set to 100, all comparisons made between the input and compare data points must return True in order for the output data point to be set to True. The comparison function selected is FN_EQ, so this means the values of the input data points must match the value of the compare data point, or the <UCPTtrueThreshold>

property if it is defined, for this to happen. If the <UCPTtrueThreshold> is defined then the value of the compare data point is not used in the comparison.

Table 50 lists several case scenarios that show when these two functions might evaluate to True.

Table 50 FN_COMPARE Examples

Input 1	**Input 2**	**Input 3**	**Input 4**	**Value of Compare Data Point**	**UCPTtrueThreshold**	**Output**
50	30	50	50	Does not matter since <UCPTtrueThreshold> is defined.	40	0.0 0
40	40	40	40	Does not matter since <UCPTtrueThreshold> is defined.	40	100.0 1
50	50	50	50	50	EMPTY	100.0 1
50	50	50	49	50	EMPTY	0.0 0

9.2.1.3 AnalogFBSet

Use the AnalogFBSet function to create new Analog Function Blocks, or to overwrite the configuration of existing Analog Function Blocks. The Analog Function Blocks to be created or written to are signified by a list of <AnalogFB> elements in the <Data> parameter. The properties that you must define within each <AnalogFB> element are the same, whether you are creating a new Analog Function Block or modifying an existing Analog Function Block. The previous section, *AnalogFBGet*, describes these properties.

NOTE: When modifying an existing Analog Function Block, any optional properties left out of the input string will be erased. Old values will not be carried over, so you must fill in every property when writing to an Analog Function Block, even if you are not changing all of the values.

You can create up to 20 Analog Function Blocks per *i.*LON 100. The AnalogFBSet function will generate the AnalogFB.XML file in the /root/config/software directory of your *i.*LON 100, if the file does not already exist.

When creating or modifying an Analog Function Block with AnalogFBSet, you may want to use output from AnalogFBGet as the basis for your <Data> parameter. You would then only need to modify the values of each property to match the new configuration you want, as opposed to re-creating an entire string like the one shown below, to generate your input.

The example below uses the AnalogFBSet function to create an Analog Function Block that calculates the maximum of the values of two input data points, NVL_nviClaValue1 and NVL_nviClaValue2, and stores the result in NVL_nviClaValue1.

<Data> Parameter

```
<Data>
   <iLONAnalogFB>
       <AnalogFB>
         <UCPTindex></UCPTindex>
         <UCPTdescription>Bielefeld AFB 1</UCPTdescription>
         <UCPTfbName></UCPTfbName>
         <UCPTcompFunction>FN_GT</UCPTcompFunction>
         <UCPTmajorityValue>100</UCPTmajorityValue>
         <UCPTtrueThreshold></UCPTtrueThreshold>
         <UCPToutputFunction>FN_MAX</UCPToutputFunction>
         <SCPTminRnge>10.0</SCPTminRnge>
         <SCPTmaxRnge>80.0</SCPTmaxRnge>
         <UCPTcalculationInterval>0.0</UCPTcalculationInterval>
         <SCPTovrBehave>OV_DEFAULT</SCPTovrBehave>
         <SCPTovrValue>0</SCPTovrValue>
         <UCPTpollOnResetDelay>0.0</UCPTpollOnResetDelay>
         <InputDataPoint>
              <Point>
                <UCPTindex>0</UCPTindex>
                <UCPTpointName>NVL_nviClaValue_1</UCPTpointName>
                <UCPTfieldName>value</UCPTfieldName>
                <UCPTpollRate>0</UCPTpollRate>
              </Point>
              <Point>
                <UCPTindex>1</UCPTindex>
                <UCPTpointName>NVL_nviClaValue_2</UCPTpointName>
                <UCPTfieldName>value</UCPTfieldName>
                <UCPTpollRate>0</UCPTpollRate>
              </Point>
         </InputDataPoint>
         <CompareDataPoint>
              <UCPTpointName>NVL_nvoClsValue_2</UCPTpointName>
```

```
                        <UCPTfieldName>value</UCPTfieldName>
                        <UCPTpollRate>0</UCPTpollRate>
                   </CompareDataPoint>
                   <OutputDataPoint>
                        <UCPTpointName>NVL_nvoClsValue_1</UCPTpointName>
                        <UCPTfieldName>value</UCPTfieldName>
                   </OutputDataPoint>
                </AnalogFB>
             </iLONAnalogFB>
          </Data>
```

<Result> Parameter

```
<Result>
  <iLONAnalogFB>
     <AnalogFB>
        <UCPTindex>1</UCPTindex>
     </AnalogFB>
   </iLONAnalogFB>
</Result>
```

9.2.1.4 AnalogFBDelete

You can use the AnalogFBDelete function to delete an Analog Function Block. You must reference the Analog Function Block to be deleted by its index number in the <Data> parameter you supply to the function, as in the example below.

<Data> Parameter

```
<Data>
  <iLON100AnalogFB>
       <AnalogFB>
         <UCPTindex>1</UCPTindex>
       </AnalogFB>
  </iLON100AnalogFB>
</Data>
```

<Result> Parameter

```
<Result>
  <iLON100AnalogFB>
       <AnalogFB>
         <UCPTindex>1</UCPTindex>
       </AnalogFB>
   </iLON100AnalogFB>
</Result>
```

10 Event Scheduler

You can use the Event Scheduler application to schedule periodic updates to the data points in your network. You will select a data point, or group of data points, for each Event Scheduler you create. These data points will be updated to specific values on the dates and times that the Event Scheduler is effective. The dates and times that the Event Scheduler is active, as well as the values the Event Scheduler will update the data points to, are completely user-defined. This section provides an overview of how this works.

Day-Based Schedules

For each event schedule, you will create up to seven day-based schedules. Each day-based schedule will apply to certain days of the week. For example, you could set up one day-based schedule that is active Monday through Friday, and another that is active Saturday and Sunday. Or, you could set up a separate day-based schedule for each day of the week.

You will define a series of day-time values for this day-based schedule, meaning that you will be allowed to specify what value you want your data points to be assigned at any given time during the days that the schedule is active. For example, you could create an Event Scheduler that sets a SNVT_switch data point to *on* (100.0 1) at 8:00 and *off* (0.0 0) at 17:00 on wekdays Monday through Friday, and leaves the data point set to *off* on Saturday and Sunday.

Date-Based Schedules

In addition, you can create date-based exceptions for each Event Scheduler. These exceptions will allow you to select specific dates which require a unique schedule, such as holidays, and assign them a schedule that is different than any of the the day-based schedules. You will be able to set up a separate set of day-time values for each exception. This allows you to specify what value you want your data points to use on each exception date at any given time, and gives you complete flexibility when creating an Event Scheduler.

The date-based exceptions must be created with the Event Calendar application. This is described in Chapter 11 of this document.

Data Points

The Event Scheduler application allows the integrator to dynamically select data points of any standard or user defined network variable type to be updated by an Event Scheduler. These outputs should be bound to network devices that require activation on a scheduled interval. The data points must be created and added to the Data Server before they are used by the Event Scheduler application. For more information on this, see *Data Server* on page 5-1.

Refreshing Exceptions

As mentioned earlier, you will use the Event Calendar application to create the exception points that define the date-based schedules for your Event Schedulers. Chapter 11 describes this procedure. The exceptions you create are stored in an exception list (a list of exceptions in UNVT_date_event format) that is stored within the Node Object. The Node Object maintains the exception list, and it receives this list via the NVL_nviDateEvent point.

All Event Schedulers on the *i.*LON 100 read the exception list from the local NodeObject internally (not with a binding), and as a result only use current exception configurations. By default, the data points of the NodeObject and the local Calendar are configured in a loop, so that this exception list comes from the local Calendar object via an internal binding between the NVL_nvoEcDateEvent output of the Calendar, and the NVL_nviDateEvent input of the NodeObject.

After a restart, the Event Scheduler recalculates the last Event Scheduler operation. It also sets the data point NVL_nvoDateResync to "100.0 1", and then to "0.0 0", which updates the *i.*LON 100 exception list. You can set the value of NVL_nvoDateResync to "100.0 1" with the DataPointWrite or DataServerWrite function at any time to refresh the exception list manually. The data point NVL_nviEcResync of the Event Calendar will be internally bound to NVL_nvoDateResync if no external binding is created. However, the Scheduler pulses the NodeObject NV to ensure that the NodeObject always has an up-to-date Exception list, so this should not be necessary.

10.1 EventScheduler.XML

The EventScheduler.XML file stores the configuration of the Event Schedulers that you have added to the *i.*LON 100.

Each Event Scheduler is signified by a <Schedule> element in the XML file. You can create Event Schedulers with the EventSchedulerSet function, or by manually editing the XML file and then downloading it to the *i.*LON 100 via FTP. The sections following this example provide instructions and guidelines to assist you when doing so.

The following represents a sample EventScheduler.XML file for an *i.*LON 100 with one defined Event Scheduler.

```
<?xml version="1.0" ?>
  <iLONEventScheduler>
    <SCPTobjMajVer>1</SCPTobjMajVer>
    <SCPTobjMinVer>1</SCPTobjMinVer>
    <Schedule>
      <UCPTindex>1</UCPTindex>
      <UCPTlastUpdate>2002-06-26T11:10:34Z</UCPTlastUpdate>
      <UCPTdescription>Office Building Control Event Scheduler</UCPTdescription>
      <UCPTfbName>Scheduler- 1</UCPTfbName>
      <UCPTscheduleEffectivePeriod>0000-00-00,0000-00-00</UCPTscheduleEffectivePeriod>
      <Point>
         <UCPTindex>0</UCPTindex>
         <UCPTpointName>NVL_nvoWeekday</UCPTpointName>
         <SCPTdelayTime>0.0</SCPTdelayTime>
      </Point>
      <Point>
         <UCPTindex>1</UCPTindex>
         <UCPTpointName>NVL_nvoWeekend</UCPTpointName>
         <SCPTdelayTime>0.0</SCPTdelayTime>
      </Point>
      <DayBased>
         <UCPTindex>0</UCPTindex>
         <UCPTdescription>Weekday</UCPTdescription>
         <UCPTpriority>240</UCPTpriority>
         <UCPTweekdays>0,1,1,1,1,1,0</UCPTweekdays>
         <DayTimeVal>
            <UCPTindex>0</UCPTindex>
            <UCPTscheduleValue>WEEKDAY</UCPTscheduleValue>
            <UCPTscheduleTime>0:0</UCPTscheduleTime>
         </DayTimeVal>
      </DayBased>
      <DayBased>
         <UCPTindex>1</UCPTindex>
         <UCPTdescription>Weekend</UCPTdescription>
         <UCPTpriority>240</UCPTpriority>
```

```
        <UCPTweekdays>1,0,0,0,0,0,1</UCPTweekdays>
        <DayTimeVal>
           <UCPTindex>0</UCPTindex>
           <UCPTscheduleValue>100.0 1</UCPTscheduleValue>
           <UCPTscheduleTime>12:30</UCPTscheduleTime>
        </DayTimeVal>
        <DayTimeVal>
           <UCPTindex>1</UCPTindex>
           <UCPTscheduleValue>0.0 0</UCPTscheduleValue>
           <UCPTscheduleTime>10:30</UCPTscheduleTime>
        </DayTimeVal>
     </DayBased>
     <DateBased>
        <UCPTindex>0</UCPTindex>
        <UCPTdescription>Datumbasierend</UCPTdescription>
        <UCPTpriority>25</UCPTpriority>
        <DateTimeVal>
           <UCPTindex>0</UCPTindex>
           <UCPTscheduleValue>OnValue</UCPTscheduleValue>
           <UCPTscheduleTime>14:0</UCPTscheduleTime>
        </DateTimeVal>
        <DateTimeVal>
           <UCPTindex>1</UCPTindex>
           <UCPTscheduleValue>Off</UCPTscheduleValue>
           <UCPTscheduleTime>15:30</UCPTscheduleTime>
        </DateTimeVal>
        <Exception>
          <UCPTindex>0</UCPTindex>
          <UCPTexceptionName>Holiday</UCPTexceptionName>
        </Exception>
     </DateBased>
   </Schedule>
</iLONEventScheduler>
```

10.2 Creating and Modifying the EventScheduler.XML File

You can create and modify the EventScheduler.XML file with the EventSchedulerSet SOAP function. The following section, *Event Scheduler SOAP Interface*, describes how to use EventSchedulerSet and the other Event Scheduler SOAP functions.

Alternatively, you can create and modify the EventScheduler.XML file manually with an XML editor, and download it to the *i.*LON 100 via FTP. Echelon does not recommend this, as the *i.*LON 100 will require a reboot to read the configuration of the downloaded file. Additionally, the *i.*LON 100 performs error checking on all SOAP messages it receives before writing to the XML file. It will not perform error checking on any XML files you download via FTP, and thus the application may not boot properly.

If you plan to create the XML file manually, you should review the rest of this chapter first, as it describes the elements and properties in the XML file that define each Event Scheduler. For instructions on creating or modifying an XML file manually, see *Manually Modifying an XML Configuration File* on page 15-1.

10.2.1 Event Scheduler SOAP Interface

The SOAP interface for the Event Scheduler application includes four functions. Table 51 lists and describes these functions. See the sections following Table 51 for more information on each function.

Table 51 Event Scheduler SOAP Functions

Function	Description
EventSchedulerList	Use this function to retrieve a list of the Event Schedulers that you have added to the *i.*LON 100. For more information, see *EventSchedulerList* on page 10-5.
EventSchedulerGet	Use this function to retrieve the configuration of an Event Scheduler. For more information, see *EventSchedulerGet* on page 10-6.
EventSchedulerSet	Use this function to create an Event Scheduler, or to modify an existing Event Scheduler. For more information, see *EventSchedulerSet* on page 10-13.
EventSchedulerDelete	Use this function to delete an Event Scheduler. For more information, see *EventSchedulerDelete* on page 10-15.

10.2.1.1 EventSchedulerList

Use the EventSchedulerList function to retrieve a list of the Event Schedulers that you have added to the *i.*LON 100. The EventSchedulerList function takes an empty string as its <Data> parameter, as in the example below.

The function returns the major and minor version numbers of the firmware implementation that the Event Scheduler application is using in the <Result> parameter. The <Result> parameter also includes a <Schedule> element for each Event Scheduler that you have added to the *i.*LON 100. The next section, *EventSchedulerGet*, describes the properties included in each of these elements.

You could use the list of <Schedule> elements returned by this function as input for the EventSchedulerGet function. The EventSchedulerGet function would then return the configuration of every Event Scheduler included in the list.

<Data> Parameter

```
Empty String
```

<Result> Parameter

```
<Result>
 <iLONEventScheduler>
   <SCPTobjMajVer>1</SCPTobjMajVer>
   <SCPTobjMinVer>1</SCPTobjMinVer>
   <Schedule>
      <UCPTindex>0</UCPTindex>
      <UCPTlastUpdate>2002-06-20T12:37:10Z</UCPTlastUpdate>
      <UCPTdescription>Office Building</UCPTdescription>
      <UCPTfbName>Scheduler- 0</UCPTfbName>
   </Schedule>
   <Schedule>
      <UCPTindex>1</UCPTindex>
      <UCPTlastUpdate>2002-06-26T11:10:34Z</UCPTlastUpdate>
      <UCPTdescription>Kitchen Schedule</UCPTdescription>
      <UCPTfbName>Scheduler- 1</UCPTfbName>
   </Schedule>
   <Schedule>
      <UCPTindex>2</UCPTindex>
      <UCPTlastUpdate>2002-06-20T12:37:11Z</UCPTlastUpdate>
      <UCPTdescription>Basement Schedule</UCPTdescription>
      <UCPTfbName>Scheduler- 2</UCPTfbName>
   </Schedule>
 </iLONEventScheduler>
</Result>
```

10.2.1.2 EventSchedulerGet

You can use the EventSchedulerGet function to return the configuration of any Event Scheduler that you have added to the *i.*LON 100. You must reference the Event Scheduler whose configuration is to be returned by its index number in the <Data> parameter you supply to the function, as in the example below.

<Data> Parameter

```
<Data>
  <iLONEventScheduler>
    <Schedule>
       <UCPTindex>1</UCPTindex>
    </Schedule>
  </iLONEventScheduler>
</Data>
```

<Result> Parameter

```
<Result>
  <iLONEventScheduler>
    <Schedule>
      <UCPTindex>1</UCPTindex>
      <UCPTlastUpdate>2002-08-26T11:10:34Z</UCPTlastUpdate>
      <UCPTdescription>Kitchen Schedule</UCPTdescription>
      <UCPTfbName>Scheduler- 1</UCPTfbName>
      <UCPTscheduleEffectivePeriod>2002-07-07,2002-08-08</UCPTscheduleEffectivePeriod>
      <Point>
        <UCPTindex>0</UCPTindex>
        <UCPTpointName>NVL_nvoWeekday</UCPTpointName>
        <SCPTdelayTime>0.0</SCPTdelayTime>
      </Point>
      <Point>
        <UCPTindex>1</UCPTindex>
        <UCPTpointName>NVL_nvoWeekend</UCPTpointName>
        <SCPTdelayTime>2.0</SCPTdelayTime>
      </Point>
      <DayBased>
        <UCPTindex>0</UCPTindex>
        <UCPTdescription>Weekday</UCPTdescription>
        <UCPTpriority>240</UCPTpriority>
        <UCPTweekdays>0,1,1,1,1,1,0</UCPTweekdays>
        <DayTimeVal>
          <UCPTindex>0</UCPTindex>
          <UCPTscheduleValue>WEEKDAY</UCPTscheduleValue>
          <UCPTscheduleTime>0:0</UCPTscheduleTime>
        </DayTimeVal>
      </DayBased>
      <DayBased>
       <UCPTindex>1</UCPTindex>
       <UCPTdescription>Weekend</UCPTdescription>
       <UCPTpriority>240</UCPTpriority>
       <UCPTweekdays>1,0,0,0,0,0,1</UCPTweekdays>
       <DayTimeVal>
         <UCPTindex>0</UCPTindex>
         <UCPTscheduleValue>100.0 1</UCPTscheduleValue>
         <UCPTscheduleTime>12:30</UCPTscheduleTime>
       </DayTimeVal>
       <DayTimeVal>
          <UCPTindex>1</UCPTindex>
          <UCPTscheduleValue>0.0 0</UCPTscheduleValue>
          <UCPTscheduleTime>10:30</UCPTscheduleTime>
        </DayTimeVal>
```

```
                  </DayBased>
                  <DateBased>
                    <UCPTindex>0</UCPTindex>
                    <UCPTdescription>Datumbasierend</UCPTdescription>
                    <UCPTpriority>25</UCPTpriority>
                    <DateTimeVal>
                      <UCPTindex>0</UCPTindex>
                      <UCPTscheduleValue>OnValue</UCPTscheduleValue>
                      <UCPTscheduleTime>14:0</UCPTscheduleTime>
                    </DateTimeVal>
                    <DateTimeVal>
                      <UCPTindex>1</UCPTindex>
                      <UCPTscheduleValue>OffValue</UCPTscheduleValue>
                      <UCPTscheduleTime>15:30</UCPTscheduleTime>
                    </DateTimeVal>
                    <Exception>
                      <UCPTindex>0</UCPTindex>
                      <UCPTexceptionName>Holiday</UCPTexceptionName>
                    </Exception>
                  </DateBased>
                </Schedule>
              </iLONEventScheduler>
            <Result>
```

The function returns a <Schedule> element for each Event Scheduler referenced in the <Data> parameter you supplied to the function. The properties included in each <Schedule> element are initially defined when the Event Scheduler is created. You can write to them with the EventCalendarSet function. Table 52 describes these properties.

Table 52 EventSchedulerGet Output Properties

Property	Description
`<UCPTindex>`	The index number assigned to the Event Scheduler must be in the range 0-32,767. As mentioned earlier, you can use the EventSchedulerSet function to create a new Event Scheduler, or to modify an existing Event Scheduler. If you do not specify an index number in the <Data> parameter you supply to EventSchedulerSet, the function will create a new Event Scheduler using the first available index number. If you specify an index number that is already being used, the function will overwrite the configuration of the Event Scheduler using that index number with the settings defined in the <Data> parameter.

Property	Description
`<UCPTlastUpdate>`	A timestamp indicating the last time the configuration of the Event Scheduler was written to. This timestamp uses the following format: YYYY-MM-DD**T**HH:MM:SS**Z** The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Event Scheduler was last updated. The second segment (**T**HH:MM:SS) represents the time of day the configuration of the Event Scheduler was last updated, in UTC (Coordinated Universal Time). UTC is the current term for what was commonly referred to as Greenwich Meridian Time (GMT). Zero (0) hours UTC is midnight in Greenwich England, which lies on the zero longitudinal meridian. Universal time is based on a 24 hour clock, therefore, an afternoon hour such as 4 pm UTC would expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002.
`<UCPTfbName>`	The functional block name assigned to the Event Scheduler in LONMAKER. You can write to this property, but each time you use the *i.*LON 100 Configuration Software to view the Event Scheduler, it will be reset to match the functional block name defined in LONMAKER.
`<UCPTdescription>`	A user-defined description of the Event Scheduler. This can be a maximum of 228 characters long.
`<UCPTscheduleEffectivePeriod>`	The period that the schedule will be effective. This date range must be entered using the following format: YYYY-MM-DD,YYYY-MM-DD. For example, if 2002-10-21,2002-10-30 is entered for this property, the schedule will be effective from 10-21-2002 to 10-30-2002. Enter 0 to not define the start or end date. For example: 0000-00-00,2003-07-02 creates an Event Scheduler that is active on all dates up to 7-2-2003. Or, 2002-08-01, 0000-00-00 creates an Event Scheduler that becomes active (permanently) on 8-1-2002. 0000-00-00,0000-00-00 creates an Event Scheduler that is always active. **NOTE:** If you use the *i.*LON 100 Configuration Software to modify the configuration of an Event Scheduler after creating it with the SOAP/XML interface, any date entered that is before 1/1/1970 will be reset to 1/1/1970. Any date entered that is after 12/31/2037 will be reset to 12/31/2037.

Property	Description
`<Point>`	The data points that will be updated by the Event Scheduler are defined by a list of <Point> elements in the <Data> parameter. For each <Point> element, you must enter the name (UCPTpointName) of the data point to be updated. In addition, you should fill in the delay time (SCPTdelayTime) property for each data point. This integer value represents the period of time, in seconds, that must elapse before this data point is updated based on a DayBased or DateBased schedule point. This allows you to stagger the updating of your data points, which may be advisable if an Event Scheduler scheduler is to update multiple data points at the same time. **NOTE:** If a SNVT_tod_event data point is used, it will only be updated if its value (current_state of next_state) has changed. If a heartbeat (UNVTminSendTime) is defined for the SNVT_tod_event data point, the time_to_next_state will be decreased with every heartbeat.
`<DayBased>`	Each Event Scheduler can have up to seven day-based schedules. These are schedules that operate based on the current day of the week. This may be useful when setting up a schedule that requires different update times for different days of the week, e.g. weekends and weekdays. The day-based schedules for your Event Scheduler are defined by a list of <DayBased> elements. For a detailed description of how to configure each <DayBased> element, see *Creating a Day-Based Schedule* on page 10-9.
`<DateBased>`	Each Event Scheduler can have one date-based schedule. You will reference the schedule exceptions created with the Event Calendar application to create this date-based schedule. The <DateBased> element defines the date-based schedule. For a detailed description of how to configure the properties and elements that define the <DateBased> element, see *Creating a Date-Based Schedule* on page 10-11.

10.2.1.2.1 Creating a Day-Based Schedule

Table 53 lists and describes the properties that should be defined within each <DayBased> element.

Table 53 Day Based Schedule Properties

Property	Description
`<UCPTindex>`	The index number of the day-based schedule.
`<UCPTdescription>`	A user-defined description of the day-based schedule. This description can be up to 228 characters long.

Property	Description
`<UCPTpriority>`	The priority assigned to the schedule, from 0 (highest priority) to 255 (lowest priority). The priority chosen here must be greater than or equal to the current priority level assigned to a data point when the Event Scheduler attempts to update that data point. If it is not, the data point will not be updated successfully. For a more detailed description of data point priority levels, see *Data Point Values and Priority Levels* on page 3-5.
`<UCPTweekdays>`	The days of the week this day-based schedule will be active, as a comma-separated list of Boolean values (1=effective, 0=not effective), starting with Saturday. For example, to create a day-based schedule that is effective on Monday and Tuesday, enter: 0,0,1,1,0,0,0 Or, to create a schedule that is effective Sunday and Wednesday, enter: 0,1,0,0,1,0,0 The default value for this property is 0,0,0,0,0,0,0.
`<DayTimeVal>`	The update events for each day-based schedule are signified by a list of <DayTimeVal> elements. Each update event will be used on the days that this day-based schedule is active. For each <DayTimeVal> element, you must enter a value, or select a value definition defined for the data points the Event Scheduler is updating, as the <UCPTscheduleValue> property. You can define value definitions for each data point with the DataServerSet function. For more information, see *DataServerSet* on page 5-10. You will also enter the local time (UCPTscheduleTime), which defines the time of day when the data points selected for your Event Scheduler will be updated to the value specified for the <UCPTscheduleValue> property. This time must be entered in 24-hour format, e.g. 15:30 represents 3:30 PM. **NOTE:** The EventScheduler application supports a maximum of 1024 update events per *i.*LON 100. As a result, the total number of <DayTimeVal> and <DateTimeVal> elements defined in the *i.*LON 100 cannot exceed 1024.

Now, consider the <DayBased> element in the sample <Data> parameter at the beginning of this section. That day-based schedule section is effective on Saturday and Sunday of the date range specified by the <UCPTscheduleEffectivePeriod> property. At 3:40 AM on every Saturday and Sunday, it updates all data points selected for the Event Scheduler to their ON value. At 6:40 PM, it updates these data points to their OFF value. This assumes that the priority level assigned to the data point being updated is between 240 and 255.

```
<DayBased>
       <UCPTindex></UCPTindex>
       <UCPTdescription>Weekend</UCPTdescription>
       <UCPTpriority>240</UCPTpriority>
       <UCPTweekdays>1,0,0,0,0,0,1</UCPTweekdays>
       <DayTimeVal>
          <UCPTindex></UCPTindex>
```

```
         <UCPTscheduleValue>ON</UCPTscheduleValue>
         <UCPTscheduleTime>03:40</UCPTscheduleTime>
      </DayTimeVal>
      <DayTimeVal>
         <UCPTindex></UCPTindex>
         <UCPTscheduleValue>OFF</UCPTscheduleValue>
         <UCPTscheduleTime>18:40</UCPTscheduleTime>
      </DayTimeVal>
   </DayBased>
```

10.2.1.2.2 Creating a Date-Based Schedule

Table 54 lists and describes the properties that should be defined within each <DateBased> element.

Table 54 Date-Based Schedule Properties

Property	Description
`<UCPTindex>`	The index number for the date-based schedule.
`<UCPTdescription>`	A user-defined description of the date-based schedule. This description can be up to 228 characters long.
`<UCPTpriority>`	The priority to be assigned the schedule, from 0 (highest priority) to 255 (lowest priority). The priority chosen here must be greater than or equal to the priority assigned to the data point when the Event Scheduler attempts to update the data point. If it is not, the data point will not be updated successfully. For a more detailed description of data point priority levels, see *Data Point Values and Priority Levels* on page 3-5.
`<Exception>`	The exceptions for the date-based schedule specify the dates on which the date-based schedule will be active. These exceptions are signified by a list of <Exception> elements. Each exception must be referenced by its name (UCPTexceptionName). You will define the name of an exception and the dates is applies to when you create it with the Event Calendar application. For more information on this, see*Event Calendar* on page 12-1.

Property	Description
`<DateTimeVal>`	The update events for each date-based schedule are signified by a list of <DateTimeVal> elements. Each update event will be used on the days that this date-based schedule is active. For each <DateTimeVal> element, you must enter a value, or select a value definition defined for the data points the Event Scheduler is updating, as the <UCPTscheduleValue> property. You can define value definitions for each data point with the DataServerSet function. For more information, see *DataServerSet* on page 5-10. You will also enter the local time (UCPTscheduleTime), which defines the time of day when the data points selected for your Event Scheduler will be updated to the value specified for the <UCPTscheduleValue> property. This time must be entered in 24-hour format, e.g. 16:30 represents 4:30 PM. **NOTE:** The Event Scheduler application supports a maximum of 1024 update events per *i.*LON 100. As a result, the total number of <DayTimeVal> and <DateTimeVal> elements defined in the *i.*LON 100 cannot exceed 1024.

Now, consider the <DateBased> element in the sample <Data> parameter at the beginning of this section. That date-based schedule section is effective on the dates assigned to the Holiday and Christmas calendar exceptions. At 3:40 AM on every date these exceptions apply to, the data points affected by this function are updated to their BYPASS values. At 11:00 PM on every date these exceptions apply to, the data points affected by this function are updated to their OFF values. This assumes that the priority level assigned to the data point being updated is between 112 and 255.

```
<DateBased0>
      <UCPTindex></UCPTindex>
      <UCPTpriority>112</UCPTpriority>
      <UCPTdescription>Exception</UCPTdescription>
      <Exception>
        <UCPTindex></UCPTindex>
        <UCPTexeptionName>Holiday</UCPTexeptionName>
      </Exception>
      <Exception>
        <UCPTindex></UCPTindex>
        <UCPTexeptionName>Christmas</UCPTexeptionName>
      </Exception>
      <DateTimeVal>
          <UCPTindex></UCPTindex>
          <UCPTscheduleValue>BYPASS</UCPTscheduleValue>
          <UCPTscheduleTime>03:40</UCPTscheduleTime>
      </DateTimeVal>
      <DateTimeVal1>
          <UCPTindex></UCPTindex>
          <UCPTscheduleValue>OFF</UCPTscheduleValue>
          <UCPTscheduleTime>23:00</UCPTscheduleTime>
      </DateTimeVal1>
    </DateBased>
```

10.2.1.3 EventSchedulerSet

You can use the EventSchedulerSet function to create new Event Schedulers, or to overwrite the configuration of existing Event Schedulers. The Event Schedulers to be created or written to are signified by a list of <Schedule> elements in the <Data> parameter. The properties you must define within each <Schedule> element are the same, whether you are creating a new Event Scheduler or modifying an existing Event Scheduler. The previous section, *EventSchedulerGet*, describes these properties.

NOTE: When modifying an existing Event Scheduler, any optional properties left out of the input string will be erased. Old values will not be carried over, so you should fill in every property when writing to an Event Scheduler, even if you are not changing all of the values.

When creating or modifying an Event Scheduler with this function, you may want to use output from EventSchedulerGet as the basis for your <Data> parameter. You would then only need to modify the values of each property to match the new configuration you want, as opposed to re-creating an entire string like the one shown below, to generate your input.

You can create up to 40 Event Schedulers per *i.*LON 100. The EventSchedulerSet function will generate the EventScheduler.XML file in the /root/config/software directory of your *i.*LON 100, if the file does not already exist.

The example below creates an Event Scheduler that will update two data points, NVL_nvoWeekend and NVL_nvoWeekday. It inlcudes two day-based schedules, one that applies to weekdays and one that applies to weekends. It also includes a date-based schedule that references a schedule exception created for holidays.

<Data> Parameter

```
<Data>
  <iLONEventScheduler>
    <Schedule>
      <UCPTindex>1</UCPTindex>
      <UCPTdescription>Event Scheduler For Office</UCPTdescription>
      <UCPTfbName>Scheduler- 1</UCPTfbName>
      <UCPTscheduleEffectivePeriod>0000-00-00,0000-00-00</UCPTscheduleEffectivePeriod>
      <Point>
        <UCPTindex>0</UCPTindex>
        <UCPTpointName>NVL_nvoWeekday</UCPTpointName>
        <SCPTdelayTime>0.0</SCPTdelayTime>
      </Point>
      <Point>
        <UCPTindex>1</UCPTindex>
        <UCPTpointName>NVL_nvoWeekend</UCPTpointName>
        <SCPTdelayTime>0.0</SCPTdelayTime>
      </Point>
      <DayBased>
        <UCPTindex>0</UCPTindex>
        <UCPTdescription>Weekday</UCPTdescription>
        <UCPTpriority>240</UCPTpriority>
        <UCPTweekdays>0,1,1,1,1,1,0</UCPTweekdays>
        <DayTimeVal>
          <UCPTindex>0</UCPTindex>
          <UCPTscheduleValue>WEEKDAY</UCPTscheduleValue>
          <UCPTscheduleTime>0:0</UCPTscheduleTime>
        </DayTimeVal>
      </DayBased>
      <DayBased>
        <UCPTindex>1</UCPTindex>
        <UCPTdescription>Weekend</UCPTdescription>
```

```
            <UCPTpriority>240</UCPTpriority>
            <UCPTweekdays>1,0,0,0,0,0,1</UCPTweekdays>
            <DayTimeVal>
                 <UCPTindex>0</UCPTindex>
                 <UCPTscheduleValue>100.0 1</UCPTscheduleValue>
                 <UCPTscheduleTime>12:30</UCPTscheduleTime>
             </DayTimeVal>
             <DayTimeVal>
                   <UCPTindex>1</UCPTindex>
                   <UCPTscheduleValue>0.0 0</UCPTscheduleValue>
                   <UCPTscheduleTime>10:30</UCPTscheduleTime>
             </DayTimeVal>
          </DayBased>
          <DateBased>
            <UCPTindex>0</UCPTindex>
            <UCPTdescription>Holiday Exceptions</UCPTdescription>
            <UCPTpriority>25</UCPTpriority>
            <DateTimeVal>
              <UCPTindex>0</UCPTindex>
              <UCPTscheduleValue>OnValue</UCPTscheduleValue>
              <UCPTscheduleTime>14:00</UCPTscheduleTime>
            </DateTimeVal>
            <DateTimeVal>
              <UCPTindex>1</UCPTindex>
              <UCPTscheduleValue>OffValue</UCPTscheduleValue>
              <UCPTscheduleTime>15:30</UCPTscheduleTime>
            </DateTimeVal>
            <Exception>
              <UCPTindex>0</UCPTindex>
              <UCPTexceptionName>Holiday</UCPTexceptionName>
            </Exception>
          </DateBased>
        </Schedule>
      </iLONEventScheduler>
    <Data>
```

<Result> Parameter

```
<Result>
  <iLONEventScheduler>
    <Schedule>
      <UCPTindex>1</UCPTindex>
    </Schedule>
  </iLONEventScheduler
</Result>
```

10.2.1.4 EventSchedulerDelete

You can use the EventSchedulerDelete function to delete an Event Scheduler. You must reference the Event Scheduler to be deleted by its index number in the <Data> parameter you supply to the function, as in the example below. You can delete more than one Event Scheduler with a single call to EventSchedulerDelete, if desired.

The following example deletes two Event Schedulers, one using index value 0 and one using index value 1.

<Data> Parameter

```
<Data>
  <iLONEventScheduler>
    <Schedule>
        <UCPTindex>0</UCPTindex>
    </Schedule>
    <Schedule>
        <UCPTindex>1</UCPTindex>
    </Schedule>
  </iLONEventScheduler>
</Data>
```

<Result> Parameter

```
<Result>
  <iLONEventScheduler>
    <Schedule>
        <UCPTindex>0</UCPTindex>
    </Schedule>
    <Schedule>
        <UCPTindex>1</UCPTindex>
    </Schedule>
  </iLONEventScheduler>
</Result>
```

11 Event Calendar

Use the Event Calendar application to define the exception points that you will reference when creating the date-based schedules for your Event Schedulers. Each exception point you create represents a date, or a group of dates. When you reference an exception in an Event Scheduler, you will be able to assign the dates for that exception a unique schedule. This may be useful when creating an Event Scheduler that requires different schedules for holidays than regular weekdays, or during different seasons of the year.

This chapter describes how to create exceptions with the Event Calendar application. Chapter 10*, Event Scheduler,* describes how to create an Event Scheduler and reference the exceptions you create.

You can create daily exceptions as one-time exceptions, or exceptions that will be repeated annually. The *i.*LON 100 supports one active Event Calendar, with up to 256 schedule exception points.

When an Event Scheduler references an exception point, the Event Calendar application supplies the dates an exception point references to the Node Object using the data point NVL_nvoEcDateEvent. The Event Scheduler then reads this exception list from the local Node Object. The information contained in the exception list includes when the exception is valid, and when the exception will recur.

Whenever an exception is modified with the functions described in this chapter, all exceptions in the Event Calendar are recalculated and copied to the NVL_nvoEcDateEvent data point as a series of updates. By default, the NVL_nvoEcDateEvent data point of the Event Calendar and the NVL_nviDateEvent data point of the Node Object are internally bound, so that no network traffic is generated. Thus, the update from the Event Calendar is passed to the local Node Object, and all the Event Schedulers will read the updated exception list from the local Node Object.

In this fashion, each Event Scheduler will always have up-to-date definitions of the exceptions it references. To force all exceptions to be recalculated and copied to the NVL_nvoEcDateEvent data point, you may update the NVL_nviEcDateResync data point (which will be internally bound to the NVL_nvoDateResync data point of the Node Object if no external binding is created) with a value of "100.0 1".

11.1 EventCalendar.XML

The EventCalendar.XML file stores the configuration of the Event Calendars that you have added to the *i.*LON 100. You can create multiple Event Calendars with up to 256 exceptions per *i.*LON 100. However, the *i.*LON 100 supports only one active Event Calendar at a time. The active Event Calendar must use index number 0.

Each defined Event Calendar is signified by a <Calendar> element in the XML file. You can create event Calendars with the EventCalendarSet function, or by manually editing the EventCalendar.XML file and downloading it to the *i.*LON 100 via FTP. The sections following this example provide instructions and guidelines to assist you when doing so.

The following represents a sample EventCalendar.XML file for an *i.*LON 100 with an Event Calendar that has three defined exceptions named Holiday, Easter, and Christmas.

```
<?xml version="1.0" ?>
  <iLONEventCalendar>
     <SCPTobjMajVer>1</SCPTobjMajVer>
     <SCPTobjMinVer>1</SCPTobjMinVer>
```

```
    <Calendar>
     <UCPTindex>0</UCPTindex>
     <UCPTlastUpdate>2002-06-26T10:44:27Z</UCPTlastUpdate>
     <UCPTdescription>Floor</UCPTdescription>
     <UCPTfbName>Calendar 1</UCPTfbName>
     <UCPTscheduleEffectivePeriod>0000-00-00,0000-00-00</UCPTscheduleEffectivePeriod>
     <Exceptions>
        <UCPTindex>0</UCPTindex>
        <UCPTexceptionName>Holiday</UCPTexceptionName>
        <UCPTtemporary>0</UCPTtemporary>
        <UCPTexceptionSchedule>0000-04-07,0000-06-07,MN_NUL,DY_NUL<UCPTexceptionSchedule>
     </Exceptions>
     <Exceptions>
        <UCPTindex>0</UCPTindex>
        <UCPTexceptionName>Christmas</UCPTexceptionName>
        <UCPTtemporary>0</UCPTtemporary>
        <UCPTexceptionSchedule>0000-24-12,0000-25-12,MN_NUL,DY_NUL<UCPTexceptionSchedule>
     </Exceptions>
     <Exceptions>
        <UCPTindex>0</UCPTindex>
        <UCPTexceptionName>Easter</UCPTexceptionName>
        <UCPTtemporary>0</UCPTtemporary>
        <UCPTexceptionSchedule>0000-4-11,0000-4-12,MN_NUL,DY_NUL<UCPTexceptionSchedule>
     </Exceptions>
   </Calendar>
</iLONEventCalendar>
```

11.2 *Creating and Modifying the EventCalendar.XML File*

You can create and modify the EventCalendar.XML file with the EventCalendarSet SOAP function. The following section, *Event Calendar SOAP Interface*, describes how to use the EventCalendarSet function and the other SOAP functions provided for the Event Calendar application.

Alternatively, you can create and modify the EventCalendar.XML file manually with an XML editor, and download it to the *i.*LON 100 via FTP. Echelon does not recommend this, as the *i.*LON 100 will require a reboot to read the configuration of the downloaded file. Additionally, the *i.*LON 100 performs error checking on all SOAP messages it receives before writing to the XML file. It will not perform error checking on any XML files you download via FTP, and thus the application may not boot properly.

If you plan to create the XML file manually, you should review the rest of this chapter first, as it describes the elements and properties in the XML file that define each Event Calendar. For instructions on creating or modifying an XML file manually, see *Manually Modifying an XML Configuration File* on page 15-1.

11.2.1 Event Calendar SOAP Interface

The SOAP interface for the Event Calendar application includes four functions. Table 55 lists and describes these functions. For more information on any of these functions, see the sections following Table 55.

Table 55 Event Calendar SOAP Interface

Function	**Description**
EventCalendarList	Use this function to retrieve a list of the Event Calendars that you have added to the *i.*LON 100. For more information, see *EventCalendarList* on page 11-4.
EventCalendarGet	Use this function to retrieve the configuration of any Event Calendar that you have added to the *i.*LON 100. For more information, see *EventCalendarGet* on page 11-5.
EventCalendarSet	Use this function to create an Event Calendar, or to overwrite the configuration of an existing Event Calendar. For more information, see *EventCalendarSet* on page 11-12.
EventCalendarDelete	Use this function to delete an Event Calendar. For more information, see *EventCalendarDelete* on page 11-13.

11.2.1.1 EventCalendarList

Use the EventCalendarList function to retrieve a list of the Event Calendars that you have added to the *i.*LON 100. The EventCalendarList function takes an empty string as its <Data> parameter, as in the example below.

The function returns the major and minor version numbers of the firmware implementation that the Event Calendar application is using in the <Result> parameter. The <Result> parameter also includes a <Calendar> element for each Event Calendar that you have added to the *i.*LON 100. The next section, *EventCalendarGet*, describes the properties included in each of these elements.

You could use the list of <Calendar> elements returned by this function as the input for the EventCalendarGet function. The EventCalendarGet function will then return the configuration of each Event Calendar included in the list.

<Data> Parameter	`Empty String`

<Result> Parameter	
	```
	<Result>
	  <iLONEventCalendar>
	    <SCPTobjMajVer>1</SCPTobjMajVer>
	    <SCPTobjMinVer>1</SCPTobjMinVer>
	    <Calendar>
	      <UCPTindex>0</UCPTindex>
	      <UCPTlastUpdate>2002-06-26T10:38:48Z</UCPTlastUpdate>
	      <UCPTdescription>Floor</UCPTdescription>
	      <UCPTfbName>Calendar- 1</UCPTfbName>
	    </Calendar>
	  </iLONEventCalendar>
	</Result>
	```

11.2.1.2 EventCalendarGet

You can use the EventCalendarGet function to return the configuration of any Event Calendar that you have added to the *i.*LON 100. You must reference the Event Calendar whose configuration is to be returned by its index number in the <Data> parameter you supply to this function, as in the example below.

<Data> Parameter

```
<Data>
  <iLONEventCalendar>
    <Calendar>
       <UCPTindex>0</UCPTindex>
    </Calendar>
  </iLONEventCalendar>
</Data>
```

<Result> Parameter

```
<Result>
  <iLONEventCalendar>
    <Calendar>
      <UCPTindex>0</UCPTindex>
      <UCPTlastUpdate>2001-06-21T20:53:21Z</UCPTlastUpdate>
      <UCPTdescription>Floor</UCPTdescription>
      <UCPTfbName>Calendar 1</UCPTfbName>
      <UCPTscheduleEffectivePeriod>00-00-00,00-00-00</UCPTscheduleEffectivePeriod>
      <Exceptions>
        <UCPTindex>0</UCPTindex>
        <UCPTexceptionName>Easter</UCPTexceptionName>
        <UCPTtemporary>0</UCPTtemporary>
        <UCPTexceptionSchedule>00-04-11,00-04-12,MN_NUL,DY_NUL</UCPTexceptionSchedule>
      </Exceptions>
    </Calendar>
  </iLONEventCalendar>
</Result>
```

The function returns a <Calendar> element for each Event Calendar referenced in the <Data> parameter you supplied to the function. The properties included in each of these elements are initially defined when the Event Calendar is created. You can write to them with the EventCalendarSet function. Table 56 describes these properties.

Table 56 EventCalendarGet Output Properties

Property	Description
`<UCPTindex>`	The index number assigned to the Event Calendar must be in the range of 0-32,767. As mentioned earlier, you can use the EventCalendarSet function to create a new Event Calendar, or to modify an existing Event Calendar. If you do not specify an index number in the <Data> parameter you supply to EventCalendarSet, the function will create a new Event Calendar using the first available index number. If you specify an index number that is already being used, the function will overwrite the configuration of the Event Calendar using that index number with the settings defined in the <Data> parameter. **NOTE:** The *i.*LON 100 supports one active Event Calendar at a time. The active Event Calendar must use index number 0.

Property	Description
`<UCPTlastUpdate>`	A timestamp indicating the last time the configuration of the Event Calendar was updated. This timestamp uses the following format: YYYY-MM-DD**T**HH:MM:SS**Z** The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Event Calendar was last updated. The second segment (**T**HH:MM:SS) represents the time of day the configuration of the Event Calendar was last updated, in UTC (Coordinated Universal Time). UTC is the current term for what was commonly referred to as Greenwich Meridian Time (GMT). Zero (0) hours UTC is midnight in Greenwich England, which lies on the zero longitudinal meridian. Universal time is based on a 24 hour clock, therefore, an afternoon hour such as 4 pm UTC would expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC.
`<UCPTdescription>`	A description of the Event Calendar. This can be a maximum of 228 characters long.
`<UCPTfbName>`	The functional block name assigned to the Event Calendar in LONMAKER. You can write to this field, but each time you use the *i.*LON 100 Configuration Software to view the Event Calendar, this property will be reset to match the functional block name defined in LONMAKER.
`<UCPTexceptionSchedule>`	This property defines the dates that the Event Calendar is effective. You must fill this property in using the following format: YYYY-MM-DD,YYYY-MM-DD The first date represents the start date, and the second date represents the end date. If the start date is undefined (0000-00-00), it means any date up to and including the end date. For example, 0000-00-00,12-12-2003 would be every date up to December 12, 2003. If the end date is undefined (0000-00-00), it means any date from the start date. For example, 12-12-2003,0000-00-00 would be any date after 2003. If both are undefined, it means the Event Calendar is always active. This is the default for the property: 0000-00-00,0000-00-00 **NOTE:** If you use the *i.*LON 100 Configuration Software to modify the configuration of an Event Calendar after creating it with the SOAP/XML interface, any date entered that is before 1/1/1970 will be reset to 1/1/1970. Any date entered that is after 12/31/2037 will be reset to 12/31/2037.
`<Exceptions>`	You can specify the dates that the Event Calendar applies to by creating exception points. The exception points that have been created for an Event Calendar are signified by a group of <Exceptions> elements. You can create up to 256 exceptions per calendar. For a description of how to configure the properties you must define within each <Exceptions > element, see the next section, *Creating an Exception Point*.

11.2.1.2.1 Creating an Exception Point

The exception points for an Event Calendar are defined by a series of <Exceptions> element. Table 57 describes the properties that must be defined within each <Exceptions> element.

Table 57 Exception Point Properties

Property	Description
`<UCPTexceptionName>`	The name of the exception point. This can be a maximum of 27 characters long. You will use this to reference the exception point from the Event Scheduler application.
`<UCPTtemporary>`	Either 0 or 1. If 0, this exception will be repeated anually. If 1, this will be a temporary exception. In this case, it will be removed from the Event Calendar, and any Event Schedulers referencing the exception, after the first time it is referenced.
`<UCPTexceptionSchedule>`	This property defines the dates that the Event Calendar is active. It uses the following format: YYYY-MM-DD, YYYY-MM-DD, DAY_T, MONTH_T The first date entered represents the date that the exception period begins. The second date entered represents the date that the exception period ends. You can set up a recurring event by setting the year, month or date for these periods to 0. The dates entered here must be within the effective period of the Event Scheduler (as defined by the UCPTexceptionSchedule property). **NOTE:** If you use the *i.*LON 100 Configuration Software to modify the configuration of an Event Calendar after creating it with the SOAP/XML interface, any date entered that is before 1/1/1970 will be reset to 1/1/1970. Any date entered that is after 12/31/2037 will be reset to 12/31/2037. DAY_T represents an identifier you can fill in to specify which days of the month the exception will be valid during the interval. For example, every third day, every fourth day, etc. Table 58 lists and defines the identifers you can use for the DAY_T field. MONTH_T represents an identifer you can fill in to specify which months the exception will be valid during the interval. Table 59 lists and defines the identifiers you can use for the MONTH_T field. A list of example strings you could use to fill in the <UCPTexceptionSchedule> property follows these tables.

Table 58 lists and describes the identifiers you can use to fill in the DAY_T field of the <UCPTexceptionSchedule> property. The exception point will active on the days specified by this property.

NOTE: If you use the *i.*LON 100 Configuration Software to modify the configuration of an Event Calendar after creating it with the SOAP/XML interface, the Configuration Software will automatically reset the Event Calendar to use the DAY_LAST_SECOND_DAY identifier.

Table 58 DAY_T Identifiers

Identifier	Description
DY_LAST_DAY_OF_MONTH	Last day of month
DY_LAST_SECOND_DAY	Second-to-last day of the month.
DY_LAST_THIRD_DAY	Third-to-last day of the month
NOTE: There are many other identifiers that use the DY_LAST_XXX_DAY format described by the last three identifiers. XXX represents an integer specifying the exact day to use, in the range of 4-30. For example, you could enter the identifer DY_LAST_20_DAY to have the exception occur on the 20th to last day of each month the exception applies to.	
DY_LAST_30_DAY	30th to last day of the month
DY_FIRST_SUN	First Sunday of each month
DY_FIRST_MON	First Monday of each month
DY_SECOND_MON	Second Monday of each month
DY_THIRD_MON	Third Monday of each month
DY_FOURTH_MON	Fourth Monday of each month
DY_FIFTH_MON	Fifth Monday of each month
DY_FIRST_SAT	First Saturday of month
DY_SECOND_SUN	Second Sunday of each month
DY_SECOND_MON	Second Monday of each month
DY_SECOND_TUES	Second Monday of each month
DY_SECOND_WED	Second Tuesday of each month
DY_SECOND_THURS	Second Wednesday of each month
DY_SECOND_FRI	Second Friday of each month
DY_SECOND_SAT	Second Saturday of each month
DY_THIRD_SUN	Third Sunday of each month
DY_THIRD_MON	Third Monday of each month
DY_THIRD_TUES	Third Monday of each month
DY_THIRD_WED	Third Tuesday of each month
DY_THIRD_THURS	Third Wednesday of each month
DY_THIRD_FRI	Third Friday of each month
DY_THIRD_SAT	Third Saturday of each month
DY_FOURTH_SUN	Fourth Sunday of each month
DY_FOURTH_MON	Fourth Monday of each month
DY_FOURTH_TUES	Fourth Monday of each month
DY_FOURTH_WED	Fourth Tuesday of each month
DY_FOURTH_THURS	Fourth Wednesday of each month

Identifier	Description
DY_FOURTH_FRI	Fourth Friday of each month
DY_FOURTH_SAT	Fourth Saturday of each month
DY _FIFTH_SUN	Fifth Sunday of each month
DY_FIFTH_MON	Fifth Monday of each month
DY_FIFTH_TUES	Fifth Tuesday of each month
DY_FIFTH_WED	Fifth Wednesday of each month
DY_FIFTH_THURS	Fifth Thursday of each month
DY_FIFTH_FRI	Fifth Friday of each month
DY_FIFTH_SAT	Fifth Saturday of each month
DY_LAST_SUN	Last Sunday of each month
DY_LAST_MON	Last Monday of each month
DY_LAST_TUES	Last Tuesday of each month
DY_LAST_WED	Last Wednesday of each month
DY_LAST_THURS	Last Thursday of each month
DY_LAST_FRI	Last Friday or each month
DY_LAST_SAT	Last Saturday of each month
DY_EVERY_SUN	Every Sunday of the date interval.
DY_EVERY_MON	Every Monday of the date interval.
DY_EVERY_TUES	Every Tuesday of the date interval.
DY_EVERY_WED	Every Wednesday of the date interval.
DY_EVERY_THURS	Every Thursday of the date interval.
DY_EVERY_FRI	Every Friday of the date interval.
DY_EVERY_SAT	Every Saturday of date interval
DY_EVERY_SECOND_DAY	Every second day of date interval
DY_EVERY_THIRD_DAY	Every third day of date interval
DY_EVERY_FOURTH_DAY	Every fourth day of the date interval
DY_EVERY_FIFTH_DAY	Every fifth day of the date interval
DY_EVERY_SIXTH_DAY	Every sixth day of the date interval
DY_NUL	Value not available

Table 59 lists and describes the identifiers you can use to fill in the MONTH_T field of the <UCPTexceptionSchedule> property. The calendar will be active during the months specified by this property.

Table 59 MONTH_T Identifiers

Identifier	Description
MN_JAN	January
MN_FEB	February
MN_MAR	March
MN_APR	April
MN_MAY	May
MN_JUN	June
MN_JUL	July
MN_AUG	August
MN_SEP	September
MN_OCT	October
MN_NOV	November
MN_DEC	December
MN_EVERY_MONTH	Every month during the interval the Event Calendar is active.
MN_EVERY_2_MONTHS	Every other month during the interval the Event Calendar is active.
MN_QUARTERLY	Every third month during the interval the Event Calendar is active.
MN_EVERY_4_MONTHS	Every fourth month during the interval the Event Calendar is active.
MN_EVERY_5_MONTHS	Every fifth month during the interval the Event Calendar is active.
MN_EVERY_6_MONTHS	Every sixth month during the interval the Event Calendar is active.
MN_EVERY_7_MONTHS	Every seventh month during the interval the Event Calendar is active.
MN_EVERY_8_MONTHS	Every eighth month during the interval the Event Calendar is active.
MN_EVERY_9_MONTHS	Every ninth month during the interval the Event Calendar is active.
MN_EVERY_10_MONTHS	Every tenth month during the interval the Event Calendar is active.
MN_EVERY_11_MONTHS	Every eleventh month during the interval the Event Calendar is active.
MN_NUL	Value not available. If this is choosen, the Event Calendar will use every month)

The following string would set up an exception that occurs every day from 2000/2/28 until 2000/3/5:

```
2000-02-28,2000-03-05,DY_NUL,MN_NUL
```

The following string would set up an exception that occurs every second day from 2000/2/28 until 2000/3/5:

```
2000-02-28,2000-03-05,DY_EVERY_SECOND,MN_NUL
```

The following string would set up an exception that occurs on the last Saturday of each month between 2000/2/26 and 2002/4/29:

```
2000-02-26,2002-04-29,DY_LAST_SAT,MN_NUL
```

The following string would set up an exception that is valid every 10th month between 1/1/2000 and 12/31/2001. This would be every day in October 2001 and every day in August 2001.

```
2000-01-01,2001-12-31,DY_NUL,MN_EVERY_10_MONTH
```

11.2.1.3 EventCalendarSet

You can use the EventCalendarSet function to create new Event Calendars, or to overwrite the configuration of existing Event Calendars. The Event Calendars to be created or written to are signified by a list of <Calendar> elements in the <Data> parameter. The properties you must define within each <Calendar> element are the same, whether you are creating a new Event Calendar or modifying an existing Event Calendar. The previous section, *EventCalendarGet*, describes these properties.

NOTE: When modifying an existing Event Calendar, any optional properties left out of the input string will be erased. Old values will not be carried over, so you must fill in every property when writing to an Event Calendar, even if you are not changing all of the values.

You can create multiple Event Calendars per *i.*LON 100. However, the *i.*LON 100 can only support one active Event Calendar at a time. The Event Calendar that is assigned index number 0 will be the active Event Calendar.

When creating or modifying an Event Calendar with EventCalendarSet, it may be useful to use output from the EventCalendarGet function as the basis for your <Data> parameter. You would then only need to modify the values of each property to match the new configuration you want, as opposed to re-creating an entire string like the one shown below, to generate your input.

The EventCalendarSet function will generate the EventCalendar.XML file in the /root/config/software directory of your *i.*LON 100, if the file does not already exist.

The following example call to the EventCalendarSet function creates an Event Calendar with two recurring exception dates: 4/11 and 4/12.

<Data> Parameter

```
<Data>
  <iLONEventCalendar>
    <Calendar>
      <UCPTindex>0</UCPTindex>
      <UCPTdescription>Floor</UCPTdescription>
      <UCPTfbName>Calendar 1</UCPTfbName>
      <Exceptions>
        <UCPTexceptionName>Easter</UCPTexceptionName>
        <UCPTtemporary>0</UCPTtemporary>
        <UCPTexceptionSchedule>0000-04-11,0000-04-12,MN_NUL,DY_NUL</UCPTexceptionSchedule>
      </Exceptions>
    </Calendar>
  </iLONEventCalendar>
</Data>
```

<Result> Parameter

```
<Result>
  <iLONEventCalendar>
    <Calendar>
      <UCPTindex>0</UCPTindex>
    </Calendar>
  </iLONEventCalendar>
</Result>
```

11.2.1.4 EventCalendarDelete

You can use the EventCalendarDelete function to delete an Event Calendar. You must reference the Event Calendar to be deleted by its index number in the <Data> parameter you supply to the function, as in the example below.

<Data> Parameter	`<Data>`
	`  <iLONEventCalendar>`
	`    <Calendar>`
	`        <UCPTindex>`**`0`**`</UCPTindex>`
	`    </Calendar>`
	`  </iLONEventCalendar>`
	`</Data>`

<Result> Parameter	`<Result>`
	`  <iLONEventCalendar>`
	`    <Calendar>`
	`      <UCPTindex>`**`0`**`</UCPTindex>`
	`    </Calendar>`
	`  </iLONEventCalendar>`
	`</Result>`

12 Type Translator

You can use Type Translators to convert data points from one network variable type to another. This may be useful when comparing data points from different vendors that use different types, and are not compatible with each other.

When creating a Type Translator, you will choose a Type Translator Rule. The Type Translator Rule defines the network variable type of the data points the Type Translator will take as input, and the network variable type it will convert these data points to. The Type Translator Rule define the scaling factors, case structures for handling enumerations and fields within structures, and offsets that will be used to determine the value to assign the output data point.

The *i.*LON 100 software includes nine pre-defined Type Translator Rules. Each one is described in detail later in this chapter. It is also possible to perform translations without using a Type Translator Rule. This is possible when converting data from one scalar type to another where no offset or multipliers are required, and when converting one type to another with the same format description.

You can convert multiple input data points to a single output data point type, or you can convert a single input data point to multiple output data points of different types using Type Translators.

You can optionally create your own Type Translator Rules, or modify the Type Translator Rules provided with the *i.*LON 100 software, using the TypeTranslatorRule SOAP functions. For more information on creating Type Translator Rules, or on modifying the Type Translator Rules provided with the *i.*LON 100 software, see *Type Translator Rules* on page 13-1.

In addition, you will specify one or more input data points, and one or more output data points. The network variable type of each data point will vary, depending on the Type Translator Rule selected. When any of the input data points are updated, the Type Translator will read the values of the input data points and assign new values to the output data points, based on the values it reads and the Type Translator Rule selected.

12.1 TypeTranslator.XML

The TypeTranslator.XML file stores the configuration of all Type Translators you have added to the *i.*LON 100.

Each defined Type Translator is signified by a <TypeTranslator> element in the XML file. You can create additional Type Translators using the TypeTranslatorSet function, or by manually editing the XML file and downloading it to the *i.*LON 100 via FTP. The sections following this example provide instructions and guidelines to follow when doing so.

The following represents a sample TypeTranslator.XML file for an *i.*LON 100 with three defined Type Translators.

```
<?xml version="1.0" ?>
  <iLONTypeTranslator>
    <SCPTobjMajVer>1</SCPTobjMajVer>
    <SCPTobjMinVer>1</SCPTobjMinVer>
    <TypeTranslator>
      <UCPTindex>1</UCPTindex>
      <UCPTlastUpdate>2002-05-14T12:42:54Z</UCPTlastUpdate>
      <UCPTdescription>Translator For SNVT_Lev_Disc</UCPTdescription>
      <UCPTfbName>Type Translator- 1</UCPTfbName>
      <UCPTtranslatorRule>SNVT_lev_disc_TO_SNVT_switch</UCPTtranslatorRule>
      <SCPTdelayTime>0.0</SCPTdelayTime>
      <InDataPoint>
         <UCPTindex>0</UCPTindex>
         <UCPTpointName>NVL_nviTransLevDisc</UCPTpointName>
      </InDataPoint>
      <OutDataPoint>
         <UCPTindex>0</UCPTindex>
         <UCPTpointName>NVL_nvoTransSwitch</UCPTpointName>
      </OutDataPoint>
    </TypeTranslator>
    <TypeTranslator>
      <UCPTindex>2</UCPTindex>
      <UCPTlastUpdate>2002-05-29T04:27:43Z</UCPTlastUpdate>
      <UCPTdescription></UCPTdescription>
      <UCPTfbName>Type Translator- 2</UCPTfbName>
      <UCPTtranslatorRule />
      <SCPTdelayTime>0.0</SCPTdelayTime>
      <InDataPoint>
         <UCPTindex>0</UCPTindex>
         <UCPTpointName>NVL_nviTransTemp_f</UCPTpointName>
      </InDataPoint>
      <OutDataPoint>
         <UCPTindex>0</UCPTindex>
         <UCPTpointName>NVL_nvoTransTemp_p</UCPTpointName>
      </OutDataPoint>
    </TypeTranslator>
    <TypeTranslator>
      <UCPTindex>3</UCPTindex>
      <UCPTlastUpdate>2002-05-14T12:42:54Z</UCPTlastUpdate>
      <UCPTdescription></UCPTdescription>
      <UCPTfbName>Type Translator- 3</UCPTfbName>
      <UCPTtranslatorRule></UCPTtranslatorRule>
      <SCPTdelayTime>0.0</SCPTdelayTime>
    <InDataPoint>
      <UCPTindex>0</UCPTindex>
      <UCPTpointName>NVL_nviTransTemp_p</UCPTpointName>
    </InDataPoint>
    <OutDataPoint>
      <UCPTindex>0</UCPTindex>
      <UCPTpointName>NVL_nvoTransTemp_f</UCPTpointName>
    </OutDataPoint>
   </TypeTranslator>
  </iLONTypeTranslator>
```

12.2 Creating and Modifying the TypeTranslator.XML File

You can create and modify the TypeTranslator.XML file with the TypeTranslatorSet SOAP function. The following section, *Type Translator SOAP Interface*, describes how to use TypeTranslatorSet and the other SOAP functions provided for the Type Translator application.

Alternatively, you can create and modify the .XML file manually with an XML editor, and download it to the *i.*LON 100 via FTP. Echelon does not recommend this, as the *i.*LON 100 will require a reboot to read the configuration of the downloaded file. Additionally, the *i.*LON 100 performs error checking on all SOAP messages it receives before writing to the XML file. It will not perform error checking on any XML files you download via FTP, and thus the application may not boot properly.

If you plan to create the XML file manually, you should review the rest of this chapter first, as it describes the elements and properties in the XML file that define each Type Translator. For instructions on creating or modifying an XML file manually, see *Manually Modifying an XML Configuration File* on page 15-1.

12.2.1 Type Translator SOAP Interface

The SOAP interface for the Type Translator application includes four functions. Table 60 lists and describes these functions. For more information on any of these functions, see the sections following Table 60.

Table 60 Type Translator SOAP Functions

Function	Description
TypeTranslatorList	Use this function to retrieve a list of the Type Translators that you have added to the *i.*LON 100. For more information, see *TypeTranslatorList* on page 12-4.
TypeTranslatorGet	Use this function to retrieve the configuration of any Type Translator that you have added to the *i.*LON 100. For more information, see *TypeTranslatorGet* on page 12-5.
TypeTranslatorSet	Use this function to create a Type Translator, or to modify an existing Type Translator. For more information, see *TypeTranslatorSet* on page 12-12.
TypeTranslatorDelete	Use this function to delete a Type Translator. For more information, see *TypeTranslatorDelete* on page 12-13.

12.2.1.1 TypeTranslatorList

Use the TypeTranslatorList function to retrieve a list of the Type Translators that you have added to the *i.*LON 100. The TypeTranslatorList function takes an empty string as its <Data> parameter, as in the example below.

The function returns the major and minor version numbers of the firmware implementation the Type Translator application is using in the <Result> parameter. The <Result> parameter also includes a <TypeTranslator> element for each Type Translator that you have added to the *i.*LON 100. The next section, *TypeTranslatorGet*, describes the properties included in each of these elements.

You could use the list of <TypeTranslator> elements as input for the TypeTranslatorGet function. The TypeTranslatorGet function would then return the configuration of each Type Translator included in the list.

<Data> Parameter

```
Empty String
```

<Result> Parameter

```
<Result>
  <iLONTypeTranslator>
       <SCPTobjMajVer>1</SCPTobjMajVer>
       <SCPTobjMinVer>1</SCPTobjMinVer>
       <TypeTranslator>
              <UCPTindex>0</UCPTindex>
              <UCPTlastUpdate>2002-05-15T09:29:20Z</UCPTlastUpdate>
              <UCPTdescription>Digital</UCPTdescription>
              <UCPTfbName>Type Translator- 0</UCPTfbName>
       </TypeTranslator>
       <TypeTranslator>
              <UCPTindex>1</UCPTindex>
              <UCPTlastUpdate>2002-05-14T12:42:54Z</UCPTlastUpdate>
              <UCPTdescription>Temperature</UCPTdescription>
              <UCPTfbName>Type Translator- 1</UCPTfbName>
       </TypeTranslator>
       <TypeTranslator>
              <UCPTindex>2</UCPTindex>
              <UCPTlastUpdate>2002-05-29T04:27:43Z</UCPTlastUpdate>
              <UCPTdescription>Energy</UCPTdescription>
              <UCPTfbName>Type Translator- 2</UCPTfbName>
       </TypeTranslator>
       <TypeTranslator>
              <UCPTindex>3</UCPTindex>
              <UCPTlastUpdate>2002-05-14T12:42:54Z</UCPTlastUpdate>
              <UCPTdescription>Lighting</UCPTdescription>
              <UCPTfbName>Type Translator- 3</UCPTfbName>
       </TypeTranslator>
  </iLONTypeTranslator>
</Result>
```

12.2.1.2 TypeTranslatorGet

You can use the TypeTranslatorGet function to retrieve the configuration of any Type Translator that you have added to the *i.*LON 100. You must reference the Type Translator whose configuration is to be returned by its index number in the <Data> parameter you supply to the function, as in the example below.

<Data> Parameter

```
<Data>
  <iLONTypeTranslator>
    <TypeTranslator>
      <UCPTindex>0</UCPTindex>
    </TypeTranslator>
   </iLONTypeTranslator>
</Data>
```

<Result> Parameter

```
<Result>
  <iLONTypeTranslator>
     <TypeTranslator>
       <UCPTindex>0</UCPTindex>
       <UCPTlastUpdate>2002-05-15T09:29:20Z</UCPTlastUpdate>
       <UCPTdescription>Translator For Two TimeStamp</UCPTdescription>
       <UCPTfbName>Type Translator- 0</UCPTfbName>
       <UCPTtranslatorRule>2xTimeStamp_to_TimeStamp</UCPTtranslatorRule>
       <SCPTdelayTime>67.0</SCPTdelayTime>
       <InDataPoint>
            <UCPTindex>0</UCPTindex>
            <UCPTpointName>NVL_nviSchedTimeSet</UCPTpointName>
       </InDataPoint>
       <InDataPoint>
            <UCPTindex>1</UCPTindex>
            <UCPTpointName>NVL_nviRtTimeDate</UCPTpointName>
       </InDataPoint>
       <OutDataPoint>
            <UCPTindex>0</UCPTindex>
            <UCPTpointName>NVL_nvoTimeSet1</UCPTpointName>
     </OutDataPoint>
   </TypeTranslator>
  </iLONTypeTranslator>
</Result>
```

The function returns one <TypeTranslator> element for each Type Translator referenced in the <Data> parameter you supplied to the function. The properties contained within each of these elements are defined when the Type Translator is created. You can write to them with the TypeTranslatorSet function. Table 61 describes these properties.

Table 61 TypeTranslatorGet Output Properties

Property	Description
`<UCPTindex>`	The index number assigned to the Type Translator must be in the range 0-32,767. As mentioned earlier, you can use the TypeTranslatorSet function to create a new Type Translator, or to modify an existing Type Translator. If you do not specify an index number in the <Data> parameter you supply to TypeTranslatorSet, the function will create a new Type Translator using the first available index number. If you specify an index number that is already being used, the function will overwrite the configuration of the Type Translator using that index number with the settings defined in the <Data> parameter.
`<UCPTlastUpdate>`	A timestamp indicating the last time the configuration of the Type Translator was updated. This timestamp uses the following format: YYYY-MM-DD**T**HH:MM:SS**Z** The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Type Translator was last updated. The second segment (**T**HH:MM:SS) represents the time of day the configuration of the Type Translator was last updated, in UTC (Coordinated Universal Time). UTC is the current term for what was commonly referred to as Greenwich Meridian Time (GMT). Zero (0) hours UTC is midnight in Greenwich England, which lies on the zero longitudinal meridian. Universal time is based on a 24 hour clock, therefore, an afternoon hour such as 4 pm UTC would expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC.
`<UCPTdescription>`	A user-defined description of the Type Translator. This can be a maximum of 228 characters.
`<UCPTfbName>`	The functional block name assigned to the Type Translator in LONMAKER. You can write to this property, but each time you use the *i.*LON 100 Configuration Software to view the Type Translator, it will be reset to match the functional block name defined in LONMAKER.
`<UCPTTranslatorRule>`	The name of the Type Translator Rule that this Type Translator will use. This determines the network variable type of the data points the Type Translator will take as input, and the network variable type that these data points will be translated to. It also determines the value to be assigned to the output data point(s) after the translation. The input and output data points you select for a Type Translator must use the network variable types specified by the Type Translator Rule. The sections immediately following this table describe the Type Translator Rules included with the *i.*LON 100 software, the identifiers you can use to reference them, and the input and output data point types you can use with them. You can also use the SOAP interface to create your own Type Translator Rules. For more information on this, see Chapter 13, *Type Translator Rules*. If no translator rule is specified, then the Type Translator will convert the input data point specified for the Type Translator to the format type of the output data point specified for the Type Translator (e.g. scalar to scalar translation with no offset and no constant, or enumeration to enumeration). In this case, the value of the output data point will be updated with the value of the input data point each time a translation is made.

Property	Description
`<SCPTdelayTime>`	This property specifies the time period to wait after any one of the Type Translator's input data points are updated before a translation will be performed, in seconds. You might consider setting this to a value greater than 0 if the Type Translator will translate multiple data points. That way, translations may only occur after most or all of the input data points have been updated. The translation will reflect any other data point updates that occur during the delay interval. If this property is set to 0, the Type Translator will perform a translation each time any of the input data points are updated.
`<InDataPoint>`	The data point, or data points, the Type Translator will translate are signified by a list of <InDataPoint> elements. Each <InDataPoint> element contains two properties: <UCPTpointName> and <UCPTindex>. Use the <UCPTpointName> property to reference the name of the input data point, as defined in the *i.*LON 100 Data Server. Use the <UCPTindex> property to assign that data point an index number within the Type Translator, if the Type Translator takes multiple input data points. The sections following this table describe the Type Translator Rules provided with the *i.*LON 100 software, and the format types that each rule requires for the input data points.
`<OutDataPoint>`	The output data point(s) that will store the translated input data point. These data points are signified by a list of <InDataPoint> elements. Each <InDataPoint> element contains two properties: <UCPTpointName> and <UCPTindex>. Use the <UCPTpointName> property to reference the name of the output data point, as defined in the *i.*LON 100 Data Server. Use the <UCPTindex> property to assign that data point an index number within the Type Translator, if the Type Translator generates multiple output data points. The sections following this table describe the Type Translator Rules provided with the *i.*LON 100 software, and the format types that each rule requires for the output data points.

12.2.1.2.1 Type Translator Rules

The following sections list the identifiers you can use to fill in the <UCPTtranslatorRule> property when creating a Type Translator. They also provide descriptions of the Type Translator Rules these identifiers reference, and of the network variable types of the input and output data points you must use with each rule.

You can find the XML files that store the configuration of these Type Translator Rules in the /root/config/Software/TranslatorRules directory of the *i.*LON 100.

12.2.1.2.1.1 16xSNVT_switch_TO_SNVT_state

Use this Type Translator Rule to convert up to 16 input data points of type SNVT_switch into an output data point of type SNVT_state. The value of the *state* field of each of the SNVT_switch input data points will be assigned to a field in the SNVT_state output data point.

The SNVT_state output data point is defined by the <OutDataPoint> element in the <Data> parameter. This element must contain the <UCPTpointName> of the SNVT_state data point that is to store the Type Translator's output.

The 16 SNVT_switch data points to be translated are defined by a list of <InDataPoint> elements. Each element must contain two properties: <UCPTpointName> and <UCPTindex>. The <UCPTpointName> must identify a SNVT_switch data point. The <UCPTindex> must be in the range 0-15.

The value of the *state* field of each input data points will be read and stored in bitX of the output data point, where X represents the <UCPTindex> selected for the input data point. For example, the *state* field of the data point assigned index number 0 in the Type Translator would be stored in Bit0 of the output SNVT_state data point. Or, the *state* field of the data point assigned index number 8 would be stored in Bit7 of the output SNVT_state data point.

If any of the index numbers for the input data points are not used (meaning that less than 16 data points were supplied to the Type Translator), then the corresponding field in the output data point will be assigned a value of 0.

12.2.1.2.1.2 SNVT_lev_disc_TO_SNVT_occupancy

Use this Type Translator Rule to translate an input data point of type SNVT_lev_disc to an output data point of type SNVT_occupancy. When you use this rule, you must reference the SNVT_lev_disc data point that is to be translated by its <UCPTpointName> in the <InDataPoint> element. You must reference the SNVT_occupancy data point to store the result of the translation by its <UCPTpointName> in the <OutDataPoint> element.

Each time a type translation is made, the Type Translator will assign the SNVT_occupancy output data point an enumeration value based on the enumeration assigned to the input data point. The enumeration values assigned to the output data point follow the rules described in Table 62.

Table 62 SNVT_lev_disc_TO_SNVT_occupancy

If the Input SNVT_lev_disc Data Point Is.....	Then the SNVT_occupancy Output Data Point Will Be Set To...
ST_NUL	OC_NUL
ST_OFF	OC_UNOCCUPIED
ST_ON	OC_OCCUPIED
ST_HIGH	OC_BYPASS
ST_LOW	OC_STANDY
ST_MED	OC_STANDY

12.2.1.2.1.3 SNVT_lev_disc_TO_SNVT_switch

Use this Type Translator Rule to translate an input data point of type SNVT_lev_disc to an output data point of type SNVT_switch. When you use this rule, you must reference the SNVT_lev_disc data point that is to be translated by its <UCPTpointName> in the <InDataPoint> element. You must reference the SNVT_switch data point to store the result of the translation by its <UCPTpointName> in the <OutDataPoint> element.

Each time a translation is made, the Type Translator will assign the SNVT_switch output data point a value that is based on the enumeration currently assigned to the input data point. The values assigned to the output data point follow the rules described in Table 63.

Table 63 SNVT_lev_disc_TO_SNVT_switch

If the Input SNVT_lev_disc Data Point Is.....	**Then the SNVT_switch Output Data Point Will Be Set To...**
ST_NUL	0xff 0
ST_OFF	0.0 0
ST_ON	100.0 1
ST_HIGH	75.0 1
ST_LOW	50.0 1
ST_MED	25.0 1

12.2.1.2.1.4 SNVT_occupancy_TO_SNVT_setting

Use this Type Translator Rule to translate an input data point of type SNVT_occupancy to an output data point of type SNVT_setting. When you use this rule, you must reference the SNVT_occupancy data point that is to be translated by its <UCPTpointName> in the <InDataPoint> element. You must reference the SNVT_setting data point that will store the result of the translation by its <UCPTpointName> in the <OutDataPoint> element.

Each time a translation is made, the three fields of the SNVT_setting data point (*function, rotation, setting)* will be assigned different values based on enumeration currently assigned to the input data point. These values assigned to these fields follow the rules described in Table 64.

Table 64 SNVT_occupancy_TO_SNVT_setting

If the SNVT_occupancy Input Data Point Is....	**Then the SNVT_setting Output Field Values Will Be Set To...**		
	function	*Setting*	*Rotation*
OC_NUL	SET_NUL	0	0.0
OC_UNOCCUPIED	SET_STATE	60	-80.01
OC_OCCUPIED	SET_STATE	100	80.24
OC_BYPASS	SET_STATE	100.0	80.24
OC_STANDBY	SET_STATE	60.2	-40.0

12.2.1.2.1.5 SNVT_scene_TO_SNVT_setting

Use this Type Translator Rule to translate an input data point of type SNVT_scene to an output data point of type SNVT_setting. When you use this rule, you must reference the SNVT_scene data point that is to be translated by its <UCPTpointName> in the <InDataPoint> element. You must reference the SNVT_setting data point to store the result of the translation by its <UCPTpointName> in the <OutDataPoint> element.

Each time a translation is made, the three fields of the SNVT_setting data output point (*function, rotation, setting)* will be assigned different values based on the values of the *function* and *scene_number* fields of the SNVT_scene data point. The values assigned to the fields of the output data point follow the rules described in Table 65.

Table 65 SNVT_scene_TO_SNVT_setting

If the SNVT_scene Input Field Values Are....		**Then the SNVT_setting Output Field Values Are....**		
function	*scene_number*	*function*	*setting*	*Rotation*
SC_RECALL	0	SET_STATE	0	0
SC_RECALL	1	SET_STATE	25	0

If the SNVT_scene Input Field Values Are....		Then the SNVT_setting Output Field Values Are....		
function	*scene_number*	*function*	*setting*	*Rotation*
SC_RECALL	2	SET_STATE	50	0
SC_RECALL	3	SET_STATE	75	0
SC_RECALL	>3	SET_NUL	100	0
SC_NUL	N/A	SET_NUL	100	0

12.2.1.2.1.6 SNVT_scene_TO_SNVT_switch

Use this Type Translator Rule to translate an input data point of type SNVT_scene to an output data point of type SNVT_switch. When you use this rule, you must reference the SNVT_scene data point that is to be translated by its <UCPTpointName> in the <InDataPoint> element. You must reference the SNVT_switch data point to store the result of the translation by its <UCPTpointName> in the <OutDataPoint> element.

Each time a translation is made, the SNVT_switch output data point will be assigned a *value* and *state* based on the values assigned to the *function* and *scene_number* fields of the SNVT_scene input data point. The value assigned to the output data point follow the rules described in Table 66.

Table 66 SNVT_scene to SNVT_switch

If the SNVT_scene Input Field Values Are....		Then the SNVT_switch Output Data Point Will Be Set To...
function	*scene_number*	
SC_NUL	N/A*	0.0 0
SC_RECALL	1	25.0 1
SC_RECALL	2	50.0 1
SC_RECALL	3	75.0 1
SC_RECALL	>3	100.0 1
SC_RECALL	255	0.0 0

*If the input *function* is SC_NUL and the input *scene_number* is 0, the value of the output data point will not be modified.

12.2.1.2.1.7 SNVT_setting_TO_SNVT_switch

Use this Type Translator Rule to translate an input data point of type SNVT_setting to an output data point of type SNVT_switch. When you use this rule, you must reference the SNVT_setting input data point that is to be translated by its <UCPTpointName> in the <InDataPoint> element. You must reference the SNVT_switch data point to store the result of the translation by its <UCPTpointName> in the <OutDataPoint> element.

Each time a translation is made, the SNVT_switch output data point will be assigned a value based on the values assigned to the *function* and *setting* fields of the SNVT_setting input data point. The value assigned to the output data point follow the rules described in Table 67.

Table 67 SNVT_setting_TO_SNVT_switch

If the SNVT_setting Input Field Values Are....		Then the SNVT_switch Output Data Point Will Be Set To...
Function	*setting*	

SET_STATE	>100.0	0xFF 0
SET_STATE	<=100.0	*(setting* value) 0
SET_NUL	N/A	0xFF 0

12.2.1.2.1.8 SNVT_state_TO_16xSNVT_switch

Use this Type Translator Rule to translate a data point of type SNVT_state to multiple output data points of type SNVT_switch. When you use this rule, you must reference the SNVT_state input data point that is to be translated by its <UCPTpointName> in the <InDataPoint> element.

The 16 SNVT_switch data points to store the result of the translation are signified by a list of <InDataPoint> elements in the <Data> parameter. Each element must contain two properties: <UCPTpointName> and <UCPTindex>. The <UCPTpointName> must identify a SNVT_switch data point. The <UCPTindex> must be in the range 0-15.

Each output data point will be assigned a value based on its index number, and the value of the corresponding field in the input data point. For example, the output data point using index number 0 within the Type Translator will be assigned a value based on *Bit0* of the input data point. The output data point using index number 7 within the Type Translator will be assigned a value based on *Bit6* of the input data point, and so on.

If the value of a BitX field is 0, then the applicable SNVT_switch data point will be assigned the value 0.0 0. If the value of a BitX field is 1, then the applicable SNVT_switch data point will be assigned the value 100.0 1.

12.2.1.2.1.9 SNVT_switch_TO_SNVT_lev_disc

Use this Type Translator Rule to translate an input data point of type SNVT_switch to an output data point of type SNVT_lev_disc. When you use this rule, you must reference the SNVT_switch input data point that is to be translated by its <UCPTpointName> in the <InDataPoint> element. You must reference the SNVT_lev_disc data point to store the result of the translation by its <UCPTpointName> in the <OutDataPoint> element.

The SNVT_lev_disc output data point will be assigned an enumeration based on the value of the *state* and *value* fields of the input data point each time a translation is made. The value assigned to the output data point follow the rules described in Table 68.

Table 68 SNVT_switch_TO_SNVT_lev_disc

If the SNVT_switch Input Field Values Are….		**Then the SNVT_lev_disc Output Data Point Will Be Set To…**
State	*value*	
0	N/A	ST_OFF
1	0.0	ST_OFF
1	1.0-25.0	ST_LOW
1	26.0-50.0	ST_MED
1	51.0-75.0	ST_HIGH
1	76.0-100.0	ST_ON
1	>100.0	ST_NUL

12.2.1.3 TypeTranslatorSet

You can use the TypeTranslatorSet function to create new Type Translators, or to overwrite the configuration of existing Type Translators. The Type Translators to be created or written to are signified by a list of <TypeTranslator> elements in the <Data> parameter. The properties you must define within each of these elements are the same, whether you are creating a new Type Translator or modifying an existing Type Translator. The previous section, *TypeTranslatorGet*, describes these properties.

NOTE: When modifying an existing Type Translator, any optional properties left out of the input string will be erased. Old values will not be carried over, so you must fill in every property when writing to a Type Translator, even if you are not changing all of the values.

When creating or modifying a Type Translator with TypeTranslatorSet, you may want to use output from TypeTranslatorGet as the basis for your <Data> parameter. You would then only need to modify the values of each property to match the new configuration you want, as opposed to re-creating an entire string like the one shown below, to generate your input.

The first invocation of the TypeTranslatorSet function will generate the TypeTranslator.XML file in the /root/config/software directory of your *i.*LON 100, if the file does not already exist.

The following uses the TypeTranslatorSet function to create a Type Translator that uses the Type Translator Rule "SNVT_switch_TO_SNVT_lev_disc" to translate the data point NVL_nviTTswitch, and store the result of the translation in the data point NVL_nvoLevDisc. Because the "SNVT_switch_TO_SNVT_lev_disc" rule is being used, NVL_nviTTswitch must be a SNVT_switch data point and NVL_nvoLevDisc must be a SNVT_lev_disc data point. The input and output data point types that must be used with the other Type Translator Rules provided with the *i.*LON 100 software are listed in the previous section, *TypeTranslatorGet.*

<Data> Parameter

```
<Data>
  <iLONTypeTranslator>
      <TypeTranslator>
        <UCPTindex></UCPTindex>
        <UCPTdescription>Translator For SNVT_switch</UCPTdescription>
        <UCPTfbName></UCPTfbName>
        <UCPTtranslatorRule>SNVT_switch_TO_SNVT_lev_disc</UCPTtranslatorRule>
        <SCPTdelayTime>0.0</SCPTdelayTime>
        <InDataPoint>
          <UCPTindex>0</UCPTindex>
          <UCPTpointName>NVL_nviTTswitch</UCPTpointName>
        </InDataPoint>
        <OutDataPoint>
          <UCPTindex>0</UCPTindex>
          <UCPTpointName>NVL_nvoLevDisc</UCPTpointName>
        </OutDataPoint>
      </TypeTranslator>
  </iLONTypeTranslator>
</Data>
```

<Result> Parameter	`<Result>`
	`  <iLONTypeTranslator>`
	`    <TypeTranslator>`
	`      <UCPTindex>`**7**`</UCPTindex>`
	`    </TypeTranslator>`
	`  </iLONTypeTranslator>`
	`</Result>`

12.2.1.4 TypeTranslatorDelete

You can use the TypeTranslatorDelete function to delete a Type Translator. You must reference the Type Translator to be deleted by its index number in the <Data> parameter you supply to the function, as in the example below.

<Data> Parameter	`<Data>`
	`  <iLONTypeTranslator>`
	`    <TypeTranslator>`
	`       <UCPTindex>`**0**`</UCPTindex>`
	`    </TypeTranslator>`
	`  </iLONTypeTranslator>`
	`</Data>`

<Result> Parameter	`<Result>`
	`    <iLONTypeTranslator>`
	`       <TypeTranslator>`
	`              <UCPTindex>`**0**`</UCPTindex>`
	`       </TypeTranslator>`
	`    </iLONTypeTranslator>`
	`</Result>`

13 Type Translator Rules

You can use the Type Translator Rule SOAP functions to create additional Type Translator Rules for the *i.*LON 100, or to modify the Type Translator Rules provided with the *i.*LON 100 software. Each Type Translator Rule defines the network variable type of the data points a Type Translator will take as input, and the network variable type these data points will be translated to. In addition, they define the factors that are required to determine the value to be assigned to the output data point during a translation, such as scaling, offsets, and case structures to handle enumerations and fields within structures. This section provides an overview of how this works.

A Type Translator referencing a Type Translator Rule will specify input data points matching the input network variable types for that rule, and output data points matching the output types for that rule. The values of the input data points will then be translated and stored in the output data points each time any of the input data points are updated.

If an input data point is a structure, you can specify which field(s) in the input data point is to be translated. Similarly, if the output data point is a structure, you can specify which field(s) the result of a translation is to be stored in. Using these features, you can configure a Type Translator Rule to convert multiple input data points into a single output data point, or a single input data point into multiple output data points.

You can optionally create case structures that define the logic for a translation. For example, if the input data point has a scalar value and the output data point is an enumeration, you could set up a case structure to map ranges of scalar values to different enumerations for the output data point. Alternatively, you could set up case rules to map the various enumeration values an input data point to scalar values, or to different enumeration values, for an output data point.

This chapter describes how to create a Type Translator Rule. Once you have created a Type Translator Rule, you can reference it from a Type Translator. For more information on the Type Translator application and how to create a Type Translator, see Chapter 12, *Type Translator*.

13.1 Type Translator Rule XML Files

The configuration of each Type Translator Rule defined for the *i.*LON 100 will be stored in an XML file in the /root/config/Software/TranslatorRules directory of the *i.*LON 100. All files in this directory are read during boot, and valid rules are made available to the Type Translator application. By default, this directory contains several XML files that you can use with your Type Translators. Chapter 13 introduces the Type Translator Rules defined by these files, and describes how to use them with a Type Translator.

This chapter describes how to use the SOAP interface to create a new Type Translator Rule, or to modify an exisiting Type Translator Rule.

The following sample shows the XML file that stores the configuration of a Type Translator Rule called 2xSwitch_to_Switch. This Type Translator Rule takes 2 SNVT_switch data points as input. It stores the *state* field of the first input data point, and the *value* field of the second input data point, in the output data point, which is also a SNVT_switch data point.

```
<TypeTranslatorRule>
    <SCPTobjMajVer>1</SCPTobjMajVer>
    <SCPTobjMinVer>1</SCPTobjMinVer>
    <UCPTlastUpdate>2002-04-05T11:12:26Z</UCPTlastUpdate>
    <UCPTdescription>Test</UCPTdescription>
    <UCPTtranslatorRule>2xSwitch_to_Switch</UCPTtranslatorRule>
    <InDataPoint>
      <UCPTindex>0</UCPTindex>
      <UCPTformatDescription>SNVT_switch</UCPTformatDescription>
    </InDataPoint>
    <InDataPoint>
      <UCPTindex>1</UCPTindex>
      <UCPTformatDescription>SNVT_switch</UCPTformatDescription>
    </InDataPoint>
    <OutDataPoint>
      <UCPTindex>0</UCPTindex>
      <UCPTformatDescription>SNVT_switch</UCPTformatDescription>
    </OutDataPoint>
    <Case>
      <UCPTindex>0</UCPTindex>
      <UCPTindexIn>0</UCPTindexIn>
      <UCPTfieldNameIn><UCPTfieldNameIn>
      <UCPTcompFct>FN_NUL</UCPTcompFct>
      <UCPTcompValue>0</UCPTcompValue>
      <Rule>
         <UCPTindex>0</UCPTindex>
         <UCPTindexIn>0</UCPTindexIn>
         <UCPTfieldNameIn>state</UCPTfieldNameIn>
         <UCPTcompFct>FN_NUL</UCPTcompFct>
         <UCPTcompValue>0</UCPTcompValue>
         <UCPTmultiplier>1</UCPTmultiplier>
         <UCPTconstant>0</UCPTconstant>
         <UCPTindexOut>0</UCPTindexOut>
         <UCPTfieldNameOut>state</UCPTfieldNameOut>
       </Rule>
       <Rule>
         <UCPTindex>1</UCPTindex>
         <UCPTindexIn>1</UCPTindexIn>
         <UCPTfieldNameIn>value</UCPTfieldNameIn>
         <UCPTcompFct>FN_NUL</UCPTcompFct>
         <UCPTcompValue>0</UCPTcompValue>
         <UCPTmultiplier>1</UCPTmultiplier>
         <UCPTconstant>0</UCPTconstant>
         <UCPTindexOut>0</UCPTindexOut>
         <UCPTfieldNameOut>value</UCPTfieldNameOut>
      </Rule>
    </Case>
  </TypeTranslatorRule>
```

13.2 Creating and Modifying the Type Translator Rule XML Files

You can create and modify the XML files for your Type Translator Rules with the TypeTranslatorRuleSet function. The following section, *Type Translator Rule SOAP Interface*, describes how to use TypeTranslatorRuleSet and the other SOAP functions provided for use with Type Translator Rules.

Alternatively, you can create the XML files for your Type Translator Rules manually, with an XML editor, and download them to the *i.*LON 100 via FTP sessions. Echelon does not recommend this, as the *i.*LON 100 will require a reboot to read the configuration of the downloaded file. Additionally, the *i.*LON 100 performs error checking on all SOAP messages it receives before writing to the file. It will not perform error checking on any XML files you download via FTP, and thus the application may not boot properly.

However, if you plan to create and manage the XML files for your Type Translator Rules manually, you should review the rest of this chapter first, as it describes the elements and properties that define each Type Translator Rule. For instructions on creating or modifying an XML file manually, see *Manually Modifying an XML Configuration File* on page 15-1.

13.2.1 Type Translator Rule SOAP Interface

The SOAP interface for the Type Translator Rule application includes four functions. Table 69 lists and describes these functions. For more information on any of these functions, see the sections following Table 69.

Table 69 Type Translator Rule SOAP Functions

Function	Description
TypeTranslatorRuleList	Use this function to retrieve a list of the Type Translators Rules that you have added to the *i.*LON 100. For more information, see *TypeTranslatorRuleList* on page 13-4.
TypeTranslatorRuleGet	Use this function to retrieve the configuration of any Type Translator Rule that you have added to the *i.*LON 100. For more information, see *TypeTranslatorRuleGet* on page 13-5.
TypeTranslatorRuleSet	Use this function to create a Type Translator Rule, or to overwrite the configuration of an existing Type Translator Rule. For more information, see *TypeTranslatorRuleSet* on page 13-12.
TypeTranslatorRuleDelete	Use this function to delete a Type Translator Rule. For more information, see *TypeTranslatorRuleDelete* on page 13-14.

13.2.1.1 TypeTranslatorRuleList

Use the TypeTranslatorRuleList function to retrieve a list of the Type Translator Rules that you have added to the *i.*LON 100. The TypeTranslatorRuleList function takes an empty string as its <Data> parameter, as in the example below.

The function returns the major and minor version numbers of the firmware implementation the Type Translator Rule application is using in the <Result> parameter. The <Result> parameter also includes a <TypeTranslatorRule> element for each TypeTranslatorRule that you have added to the *i.*LON 100. The next section, *TypeTranslatorRuleGet*, describes the properties included in each of these elements.

You could use the list of <TypeTranslatorRule> elements returned by this function as input for the TypeTranslatorRuleGet function. The TypeTranslatorRuleGet function would then return the complete configuration of each Type Translator Rule included in the list.

<Data> Parameter

```
Empty String
```

<Result> Parameter

```
<Result>
  <iLONTypeTranslatorRule>
    <SCPTobjMajVer>1</SCPTobjMajVer>
    <SCPTobjMinVer>1</SCPTobjMinVer>
    <TypeTranslatorRule>
       <UCPTindex>7</UCPTindex>
       <UCPTlastUpdate>2002-01-30T16:32:26Z</UCPTlastUpdate>
       <UCPTtranslatorRule>SNVT_state_TO_16xswitch</UCPTtranslatorRule>
       <UCPTdescription>Converts SNVT_state to 16 different SNVT_switch </UCPTdescription>
    </TypeTranslatorRule>
    <TypeTranslatorRule>
       <UCPTindex>8</UCPTindex>
       <UCPTlastUpdate>2002-01-30T16:32:26Z</UCPTlastUpdate>
       <UCPTtranslatorRule>SNVT_switch_TO_SNVT_lev_disc</UCPTtranslatorRule>
       <UCPTdescription>Converts SNVT_switch to SNVT_lev_disc</UCPTdescription>
    </TypeTranslatorRule>
    <TypeTranslatorRule>
       <UCPTindex>9</UCPTindex>
       <UCPTlastUpdate>2002-05-22T09:33:44Z</UCPTlastUpdate>
       <UCPTtranslatorRule>2xTimeStamp_to_TimeStamp</UCPTtranslatorRule>
       <UCPTdescription>Test</UCPTdescription>
    </TypeTranslatorRule>
    <TypeTranslatorRule>
       <UCPTindex>10</UCPTindex>
       <UCPTlastUpdate>2002-01-30T16:32:26Z</UCPTlastUpdate>
       <UCPTtranslatorRule>Limit_SNVT_count_f</UCPTtranslatorRule>
       <UCPTdescription>Sets SNVT_count_f to 0 when 60000 increased</UCPTdescription>
    </TypeTranslatorRule>
  </iLONTypeTranslatorRule>
</Result>
```

13.2.1.2 TypeTranslatorRuleGet

You can use the TypeTranslatorGet function to return the configuration of any Type Translator Rule that you have added to the *i.*LON 100. You must reference the Type Translator Rule whose configuration is to be returned by its index number in the <Data> parameter you supply to the function, as in the example below.

<Data> Parameter

```
<Data>
   <iLONTypeTranslatorRule>
      <TypeTranslatorRule>
         <UCPTindex>10</UCPTindex>
      </TypeTranslatorRule>
   </iLONTypeTranslatorRule>
</Data>
```

<Result> Parameter

```
<Result>
   <iLONTypeTranslatorRule>
       <TypeTranslatorRule>
          <UCPTindex>10</UCPTindex>
          <UCPTlastUpdate>2004-05-26T01:10:35Z</UCPTlastUpdate>
          <UCPTdescription>Adds five degrees to the temperature</UCPTdescription>
          <UCPTtranslatorRule>temp_add05</UCPTtranslatorRule>
          <InDataPoint>
              <UCPTindex>0</UCPTindex>
              <UCPTformatDescription>SNVT_temp_f</UCPTformatDescription>
          </InDataPoint>
          <OutDataPoint>
              <UCPTindex>0</UCPTindex>
              <UCPTformatDescription>SNVT_temp_f</UCPTformatDescription>
          </OutDataPoint>
          <Case>
              <UCPTindex>0</UCPTindex>
              <UCPTfieldNameIn></UCPTfieldNameIn>
              <UCPTindexIn>0</UCPTindexIn>
              <UCPTcompFunction>FN_NUL</UCPTcompFunction>
              <UCPTcompValue>0</UCPTcompValue>
             <Rule>
                <UCPTindex>0</UCPTindex>
                <UCPTindexIn>0</UCPTindexIn>
                <UCPTfieldNameIn></UCPTfieldNameIn>
                <UCPTindexOut>0</UCPTindexOut>
                <UCPTfieldNameOut></UCPTfieldNameOut>
                <UCPTcompFunction>FN_NUL</UCPTcompFunction>
                <UCPTcompValue>0</UCPTcompValue>
                <UCPTmultiplier>1</UCPTmultiplier>
                <UCPTconstant>5.0</UCPTconstant>
             </Rule>
          </Case>
       </TypeTranslatorRule>
</iLONTypeTranslatorRule>
</Result>
```

The function returns one <Rule> element for each Type Translator Rule referenced in the <Data> parameter you supplied to the function. The properties and elements contained within each <Rule> element are defined when the Type Translator Rule is created. You can write to them using the TypeTranslatorRuleSet function. Table 70 describes these properties.

Table 70 TypeTranslatorRuleGet Properties

Property	Description
`<UCPTindex>`	The index number assigned to the Type Translator Rule must be in the range 0-32,767. As mentioned earlier, you can use the TypeTranslatorRuleSet function to create a new Type Translator Rule, or to modify an existing Type Translator Rule. If you do not specify an index number in the <Data> parameter you supply to TypeTranslatorRuleSet, the function will create a new Type Translator Rule using the first available index number. If you specify an index number that is already being used, the function will overwrite the configuration of the Type Translator Rule using that index number with the settings defined in the <Data> parameter.
`<UCPTlastUpdate>`	A timestamp indicating the last time the configuration of the Type Translator Rule was updated. This timestamp uses the following format: YYYY-MM-DD**T**HH:MM:SS**Z** The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Type Translator Rule was last updated. The second segment (**T**HH:MM:SS) represents the time of day the configuration of the Type Translator Rule was last updated, in UTC (Coordinated Universal Time). UTC is the current term for what was commonly referred to as Greenwich Meridian Time (GMT). Zero (0) hours UTC is midnight in Greenwich England, which lies on the zero longitudinal meridian. Universal time is based on a 24 hour clock, therefore, an afternoon hour such as 4 pm UTC would expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC.
`<UCPTdescription>`	A description of the Type Translator Rule. This can be a maximum of 228 characters long.
`<UCPTtranslatorRule>`	The name of the Type Translator Rule. The XML file created for this Type Translator Rule will use this as its file name. For example, the XML file for the rule defined in the sample <Data> parameter shown above would be: temp_add05.XML. The name can be a maximum of 65 characters long. You will use it to reference the rule when creating a Type Translator with the TypeTranslatorSet function. For more information on the TypeTranslatorSet function, see Chapter 12, *Type Translator*. The following characters are restricted: `/ \ : * ? " < > * |`

Property	Description
`<InDataPoint>`	You can define the network variable types a Type Translator Rule accepts as input with a series of <InDataPoint> elements. Each <InDataPoint> element must contain two properties: <UCPTindex> and <UCPTformatDescription>. Use the <UCPTformatDescription> property to define the network variable type. Use the <UCPTindex> to assign that type an index number to be used within the Type Translator Rule. When you create a Type Translator to use a rule, you will define an input data point, or a group of input data points, and assign each data point an index number. The <UCPTformatDescription> of each data point must match the <UCPTformatDescription> of the <InDataPoint> element using the same index number within the Type Translator Rule. Otherwise, an error may occur during translation.
`<OutDataPoint>`	You can define the network variable types a Type Translator Rule will translate its input to with a series of <OutDataPoint> elements. Each <OutDataPoint> element must contain two properties: <UCPTindex> and <UCPTformatDescription>. Use the <UCPTformatDescription> property to define the network variable type. Use the <UCPTindex> to assign that type an index number to be used within the Type Translator Rule. When you create a Type Translator that uses a rule, you will define an output data point, or a group of output data points, and assign each data point an index number. The <UCPTformatDescription> of each data point must match the <UCPTformatDescription> of the <InDataPoint> element using the same index number within the Type Translator Rule. Otherwise, an error may occur during translation.
`<Case>`	The input and output network variable types for a Type Translator Rule are defined by a series of <InDataPoint> and <OutDataPoint> elements. You can create case structures to determine the values that will be assigned to the output data points when translations are made. This may be useful when converting scalar values to enumerations, and vice versa. The case structures for a Type Translator Rule are defined by a list of <Case> elements. For more information on case structures, see the next section, *Creating a Case Structure*.

13.2.1.2.1 Creating a Case Structure

You can create case structures for each Type Translator Rule that define the set of operations that will be performed when a Type Translator using that rule makes a translation. Each case structure includes several global elements, and a series of case rules. The case rules are signified by a list of <Rule> elements. You can use these rules to establish the value that will be assigned to the data point that the Type Translator Rule generates as output.

Before the operations defined by the case rules are performed, the Type Translator Rule will use its global elements to compare the value of an input data point (and field, where applicable) to a value of your choice. You will select a comparison function with which the comparison is to be made.

If the result of the operation is True, each of the case rules defined for the case structure will be used. If the result is False, the case rules will not be used. These comparisons are meant to give you flexibility when setting up your case structures.

For example, consider a case where the input data point for a Type Translator Rule uses the format type SNVT_occupancy. You could set up one case structure to be used when the data point is set to OC_OCCUPIED. You could set up another case structure to be used when the data point is set to OC_UNOCCUPIED. Each structure could have a different set of case rules that will be used to assign the output data point, or data points, a different value.

NOTE: If none of the case structures for a Type Translator Rule evaluate to True, the data point will be updated during the translation. However, its value will not change.

Table 71 describes the global elements you will fill in to define the comparison that will be performed. These elements must be inserted at the top of the case structure, before the <Rule> elements.

Table 71 Case Structure Global Properties

Property	Description
`<UCPTindex>`	The index number of the case structure.
`<UCPTfieldNameIn>`	If the input data point to be used in the comparison for this case structure is a structure, enter the name of the field whose value is to be used in the comparison. Leave this property empty if the input data point is not a structure.
`<UCPTindexIn>`	The index number of the input data point whose value you want to be used in the comparison, as defined within the <InDataPoint> elements of the Type Translator Rule.

`<UCPTcompFunction>`
 	
`<UCPTcompValue>`
	Select a comparison function (UCPTcompFunction) and compare value (UCPTcompValue) for the case element. Table 72 lists and describes the comparison functions that can be used to fill in the <UCPTcompFunction> property.
	
	The value of the input data point, or input data point field, selected for the case structrure will be compared to the compare value using the selected comparison function. If the result of this comparison is True, the case rules defined for the case structure will be used.
	
	For more information information on case rules, see *Case Rules* on page 13-9.

Table 72 lists and describes the comparison functions that can be used to fill in the <UCPTcompFunction> property. Each function must be referenced by the identifier string listed in the table.

Table 72 Comparison Function Identifiers

Identifier	Description
`FN_GT`	Greater than. Returns True if the value of the input data point is greater than that of the compare data point.

Identifier	Description
`FN_LT`	Less than. Returns True if the value of the input data point is less than that of the compare data point.
`FN_GE`	Greater than or equal to. Returns True if the value of the input data point is greater than or equal to that of the compare data point.
`FN_LE`	Less than or equal to. Returns True if the value of the input data point is less than or equal to that of the compare data point.
`FN_EQ`	Equal. Returns True if the value of the input data point is equal to that of the compare data point.
`FN_NE`	Not equal. Returns True if the value of the input data point is not equal to that of the compare data point.
`FN_NUL`	Null. Returns True for all values of the input. Use this if you want the case rules for a structure to be used each time there is a translation.

13.2.1.2.1.1 Case Rules

You can use case rules to determine the value(s) to be assigned to the output data point(s) when a Type Translator Rule is used. If the output data point is a structure, you can create case rules to determine the value that will be assigned to each field in the structure.

For each case rule, you will specify an input data point (and a field name if the input data point is a structure) to determine the input value. You will also specify a compare value and a comparison function. The input value will be compared to the compare value using the specified comparison function. If the result of the comparison is True, the operation defined by the case rule will be performed. If the result of the comparison is False, the operation will not be performed and the value of the output data point (or field) will not change.

Consider a Type Translator Rule that converts a SNVT_scene data point into a SNVT_switch data point. You could create a case rule to assign the *value* or *state* fields of the SNVT_switch data point a value based on *scene_number* of the SNVT_scene data point. For example, you could assign the SNVT_switch data point the value 100.0 1 if the *scene_number* is less than 2, or 0.0 if it is greater than 2. You can create as many case rules as you want per case structure, so you can plan on as many contingencies as you like.

Each case rule is defined by a <Rule> element. Table 73 describes the properties that should be filled in within each <Rule> element to define each case rule.

Table 73 Case Rule Properties

Property	Description
`<UCPTindex>`	The index number of the case rule. **NOTE:** If more than one case rule attempts to assign a value to the same data point or data point field, the case rule listed last in the XML file will take precedence.

Property	Description
`<UCPTindexIn>`	The index number of the input data point you want the case rule to use, as defined within the <InDataPoint> elements of the Type Translator Rule. The value of this data point will be compared to the <UCPTcompValue> selected for the case rule using the comparison function defined by the <UCPTcompFunction> property. If the result of the comparison is True, the case rule will modify the value of the input data point using the operations determined by the <UCPTmultiplier> and <UCPTconstant> properties, and assign the resulting value to the output data point chosen for the case rule.
`<UCPTfieldNameIn>`	If the input data point for this Type Translator Rule is a structure, enter the name of the field from which the input value for the case rule should be taken. This can be a maximum of 31 characters. Leave this property blank if the input data point is not a structure.
`<UCPTindexOut>`	The index number of the output data point to store the value calculated by this case rule, as defined within the <OutDataPoint> elements of the Type Translator Rule.
`<UCPTfieldNameOut>`	If the output data point chosen for this case rule is a structure, enter the name of the field in the output data point to store the result of this calculation. This can be a maximum of 31 characters. Leave this property blank if the output data point is not a structure.

`<UCPTcompFunction>`
 	
`<UCPTcompValue>`
	Select a comparison function (UCPTcompFunction) and comparison value (UCPTcompValue) for the case rule. The <UCPTcompValue> selected must use the same value format as the input data point, or field, selected for the case rule. Table 72 lists and describes the comparison functions you can use to fill in the <UCPTcompFunction> property.
	
	The value of the input data point, or input data point field, will be compared to the compare value using the compare function selected here. If the result of the comparison is True, the operation defined by the <UCPTmultiplier> and <UCPTconstant> properties will be performed. If the result of the comparison is False, the operation will not be performed and the value of the output data point (or field) will not change.

Property	Description
`<UCPTmultiplier>`	If the output data point, or data point field, takes a numeric value as its value type, enter a numeric value here. The Type Translator will multiply the value of the input data point, or data point field, for the case rule by this number and store the resulting value in the output data point (field) if the comparison for the case rule evaluates as True. You can use the <UCPTconstant> field to add a sum to this value after the multiplication has been performed. If the output data point takes an enumeration as its value, leave this property empty.
`<UCPTconstant>`	If the output data point, or data point field, takes an enumeration as its value type, enter the enumeration the output data point is to be assigned when the comparison for the case rule evaluates to True. If the output data point, or data point field, takes a numeric value as its value type, enter a numeric value here. The Type Translator will be add this to the value of the input data point, or data point field, for the case rule and store the resulting sum in the output data point (field). This Type Translator will perform this operation after the multiplication operation defined by the <UCPTmultiplier> property is performed.

13.2.1.3 TypeTranslatorRuleSet

Use the TypeTranslatorRuleSet function to create a new Type Translator Rule, or to overwrite the configuration of an exisiting Type Translator Rule. Each time you use this function to create a new Type Translator Rule, an XML file for that rule will be generated in the /root/config/software/TranslatorRules directory of your *i.*LON 100. Once the file has been generated, you can reference the rule when creating a Type Translator, as described in Chapter 12.

The previous section, *TypeTranslatorRuleGet*, describes the properties and elements you can use to define each Type Translator Rule.

The following example uses the TypeTranslatorRuleSet function to create a Type Translator Rule defintion that will convert data points of type SNVT_lev_disc to data points of type SNVT_switch. The rule takes a single data point as input, and returns a single data point as output.

NOTE: Type Translator Rules created with the TypeTranslatorRuleSet function are not supported by the *i.*LON 100 Configuration Software.

<Data> Parameter

```
<Data>
  <iLONTypeTranslatorRule>
  <TypeTranslatorRule>
    <UCPTindex></UCPTindex>
    <UCPTdescription>Converts SNVT_lev_disc to SNVT_occupancy </UCPTdescription>
    <UCPTtranslatorRule>SNVT_lev_disc_TO_SNVT_occupancy</UCPTtranslatorRule>
    <InDataPoint>
       <UCPTindex>0</UCPTindex>
       <UCPTformatDescription>SNVT_lev_disc</UCPTformatDescription>
       </InDataPoint>
       <OutDataPoint>
          <UCPTindex>0</UCPTindex>
          <UCPTformatDescription>SNVT_occupancy</UCPTformatDescription>
       </OutDataPoint>
       <Case>
          <UCPTindex>0</UCPTindex>
          <UCPTfieldNameIn></UCPTfieldNameIn>
          <UCPTindexIn>0</UCPTindexIn>
          <UCPTcompFunction>FN_NUL</UCPTcompFunction>
          <UCPTcompValue>0</UCPTcompValue>
          <Rule>
             <UCPTindex>0</UCPTindex>
             <UCPTindexIn>0</UCPTindexIn>
             <UCPTfieldNameIn></UCPTfieldNameIn>
             <UCPTindexOut>0</UCPTindexOut>
             <UCPTfieldNameOut></UCPTfieldNameOut>
             <UCPTcompFunction>FN_EQ</UCPTcompFunction>
             <UCPTcompValue>ST_NUL</UCPTcompValue>
             <UCPTmultiplier>0</UCPTmultiplier>
             <UCPTconstant>OC_NUL</UCPTconstant>
          </Rule>
          <Rule>
             <UCPTindex>1</UCPTindex>
             <UCPTindexIn>0</UCPTindexIn>
             <UCPTfieldNameIn></UCPTfieldNameIn>
             <UCPTindexOut>0</UCPTindexOut>
             <UCPTfieldNameOut></UCPTfieldNameOut>
             <UCPTcompFunction>FN_EQ</UCPTcompFunction>
             <UCPTcompValue>ST_OFF</UCPTcompValue>
```

```
                    <UCPTmultiplier>0</UCPTmultiplier>
                    <UCPTconstant>OC_UNOCCUPIED</UCPTconstant>
                 </Rule>
                 <Rule>
                    <UCPTindex>2</UCPTindex>
                    <UCPTindexIn>0</UCPTindexIn>
                    <UCPTfieldNameIn></UCPTfieldNameIn>
                    <UCPTindexOut>0</UCPTindexOut>
                    <UCPTfieldNameOut></UCPTfieldNameOut>
                    <UCPTcompFunction>FN_EQ</UCPTcompFunction>
                    <UCPTcompValue>ST_ON</UCPTcompValue>
                    <UCPTmultiplier>0</UCPTmultiplier>
                    <UCPTconstant>OC_OCCUPIED</UCPTconstant>
                 </Rule>
                 <Rule>
                    <UCPTindex>3</UCPTindex>
                    <UCPTindexIn>0</UCPTindexIn>
                    <UCPTfieldNameIn></UCPTfieldNameIn>
                    <UCPTindexOut>0</UCPTindexOut>
                    <UCPTfieldNameOut></UCPTfieldNameOut>
                    <UCPTcompFunction>FN_EQ</UCPTcompFunction>
                    <UCPTcompValue>ST_HIGH</UCPTcompValue>
                    <UCPTmultiplier>0</UCPTmultiplier>
                    <UCPTconstant>OC_BYPASS</UCPTconstant>
                 </Rule>
              </Case>
           </TypeTranslatorRule>
         </iLONTypeTranslatorRule>
       </Data>
```

<Result> Parameter

```
<Result>
  <iLONTypeTranslatorRule>
    <TypeTranslatorRule>
       <UCPTindex>13</UCPTindex>
    </TypeTranslatorRule>
  </iLONTypeTranslatorRule>
</Result>
```

13.2.1.4 TypeTranslatorRuleDelete

You can use the TypeTranslatorRuleDelete function to delete a Type Translator Rule. You must reference the Type Translator Rule to be deleted by its index number in the <Data> parameter you supply to the function, as in the example below.

<Data> Parameter

```
<Data>
   <iLONTypeTranslatorRule>
      <TypeTranslatorRule>
         <UCPTindex>10</UCPTindex>
      </TypeTranslatorRule>
   </iLONTypeTranslatorRule>
</Data>
```

<Result> Parameter

```
<Result>
   <iLONTypeTranslatorRule>
      <TypeTranslatorRule>
         <UCPTindex>10</UCPTindex>
      </TypeTranslatorRule>
   </iLONTypeTranslatorRule>
</Result>
```

14 Using the SOAP Interface as a Web Service

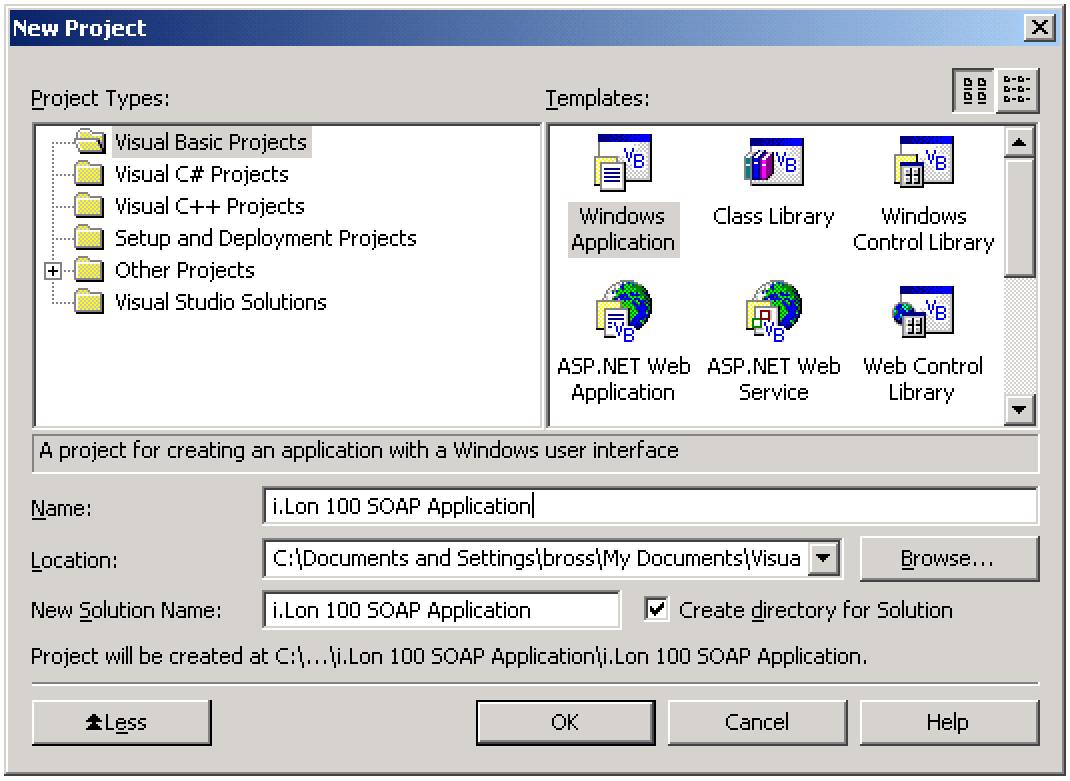
This chapter assumes that you have some familiarity with *Web services* programming, and are using the Microsoft Visual Studio .NET development environment. All sample code in this chapter is provided in Visual Basic .NET. However, you can use any development tool that is able to call standard Web services with the *i.*LON 100 SOAP interface.

14.1 Referencing the WSDL File

You can use the SOAP interface as a Web reference using Microsoft Visual Studio .NET, and create an application to modify the configuration of your *i.*LON 100 using the SOAP functions. When writing such applications, some tools can import the *i.*LON 100 WSDL file and automatically build a class structure for sending and receiving each message. The following procedure describes how to do so in Visual Basic .NET.

1. Open the Microsoft Visual Basic .NET development environment.

2. From the **File** menu, select **New**. This opens the dialog box shown in Figure 15.1.



A project for creating an application with a Windows user interface

Name: i.Lon 100 SOAP Application

Location: C:\Documents and Settings\bross\My Documents\Visua

New Solution Name: i.Lon 100 SOAP Application

☑ Create directory for Solution

Project will be created at C:\...\i.Lon 100 SOAP Application\i.Lon 100 SOAP Application.

Figure 15.1 **Create Windows Application**

3. Enter a name and select a location for the project. Click **OK**.

4. In order to take advantage of the latest features of the *i.*LON 100 SOAP/XML interface, add a Web reference to the version 1.1 *i.*LON 100 WSDL file to your project. From the **Project** menu, select **Add Web Reference**. This opens the Add Web Reference window. Enter the following in the Address text box at the top of the window:

http://[ilon100 ipaddress]/WSDL/v1.1/iLON100.WSDL [ilon 100 ipaddress] represents the IP address assigned to the *i.*LON 100 you want the application to reference.

5. Press **Enter** to download the web reference, then click **Add Reference**. The new Web reference will appear in the list of references in the Solution Explorer.

6. Create an instance of the Web service in your project form. You can do so by adding the following lines of code to your project:

```
Dim [iLonWebService] As New [name].iLON100()
```

[name] represents the name assigned to the Web reference in the Solution Explorer. [iLonWebService] represents the instance name assigned to the Web service.

5. You can now use the [iLonWebService] instance to use any of the function provided in the SOAP interface. The following section, *Programming Samples*, provides code samples to assist you in this.

14.2 Programming Samples

This section contains programming samples written in Visual Basic .NET that you may find useful when creating applications that use the SOAP interface. As described in Chapters 4-13 of this document, the input data supplied to each of the *i*.LON 100 SOAP functions for the Data Server, Data Logger, Alarm Generator, Alarm Notifier, Analog Function Block, Event Scheduler, Event Calendar, Type Translator application is a string of encoded XML containing a list of objects and values.

This string must be passed to the *i*.LON 100 within the <Data> parameter of the SOAP message the function generates. Visual Basic .NET provides several ways to generate this string and pass it to the <Data> parameter.

As noted, the programming examples in this section are written in Visual Basic .NET. For example applications written for other SOAP developmenent environments, please go to the *i*.LON Product family Web page on Echelon's Web site at http://www.echelon.com/ilon and select the "*i*.LON How-Tos and Other Internet Connectivity Resources" link to enter the *i*.LON Products How-Tos and Other Internet Connectivity Resources Web page. From there, select the *i*.LON 100 Code Examples link to access the other example applications.

NOTE: As discussed in Chapter 4, the contents of the <Data> parameter must be passed over the network in encoded XML format. Some development environments, such as Microsoft Visual Basic .NET, accept data in standard format and converts it to encoded XML for you. However, not all development environments do so. For more information on encoded XML and standard XML, see *Encoded XML and Standard XML* on page 4-6.

14.2.1 Manually Creating the <Data> Parameter

One way to generate the XML string containing the input data for a function and pass it to the <Data> parameter is to manually create a static XML string, as in the example below. This example includes comments explaining each step of this process.

```
Dim myWebReference As New iLON100_Reference.iLON100()
Dim myXMLInputString As String

'Manually create the XML string to be contained within the <Data>
'parameter when DataServerRead is called and the SOAP message is
'sent.

myXMLInputString = "<iLONDataServer><UCPTdataPointType>NVL _
    </UCPTdataPointType><UCPTstartIndex>0</UCPTstartIndex> _
    <UCPTcount>50</UCPTcount></iLONDataServer>"

'Specify the IP location of the i.LON 100 by replacing the default
' "localhost" with the IP address used by the i.LON 100.
Dim myURL As String
myURL = myWebReference.Url()
myWebReference.Url = myURL.Replace("localhost", "192.168.1.253")

'You should parse the XML string returned by the function either
'manually or with an XML parser before displaying it. The
'programming samples included later in this chapter show how you
'might do this.

Dim myXMLReturnString As String
```

```
myXMLReturnString = myWebReference.DataServerRead(myXMLInputString)
```

14.2.2 Using an XMLDocument Object

Visual Basic .NET provides several features you may find useful when programming with XML. One of these features is an object type called an XMLDocument. This section provides two examples of how you might use an XMLDocument object to parse and manipulate the XML data returned by the SOAP functions.

14.2.2.1 Writing to a Data Point

The following example uses an XMLDocument object to construct the <Data> parameter to be passed to the SOAP functions. It also uses the List/Get/Set algorithm described in Chapter 4 of this document to write to the configuration of a data point. This algorithm is very useful when writing to the configuration of an item, as it does not require you to manually create the XML string to be supplied to the Set function. Instead, you modify the output from the Get function to serve as the input. For a general description of the List/Get/Set algorithm, see *List, Get, Set and Delete Functions* on page 4-10.

This sample invokes the DataServerList to retrieve a list containing the index numbers of all data points that have been added to the *i.*LON 100 Data Server. It then uses the DataServerGet function to retrieve the configuration of the data point that has been assigned index number 0.

The DataServerGet function returns an XML string containing the configuration of the selected data point. This sample uses an XMLDocument object to store the configuration returned by DataServerGet, and to modify the value of the <UCPTlocation> property within that string. It then passes the string to the DataServerSet function as input. Once DataServerSet is invoked, the <UCPTlocation> of the selected data point will be updated in the DataServer.

You could use this algorithm to write to any number of properties for any number of applications, without having to create an entire input string for the applicable Set function. You could also use it to create new objects for applications, by modifying or removing the index number returned by the Get function before passing it to Set.

```
'Declare an instance of the i.LON 100 Web Service.
Dim myWebReference As New iLON100_Reference.iLON100()

'Specify IP address of the i.LON 100 by replacing the default
'"localhost" with the IP address used by the i.LON 100.

Dim myURL As String
myURL = myWebReference.Url()
myWebReference.Url = myURL.Replace("localhost", "10.5.250.59")

'Call DataServerList and store the <Result> parameter retrieved by
'the function in the String variable myXMLreturnString.

Dim myXMLreturnString As String
myXMLreturnString = myWebReference.DataServerList("")

'Instantiate an XMLDocument object to parse the <Result> element
'returned by the DataServerList function. This allows the data in
```

```
        'the string to be retrieved and manipulated. Load the <Result>
        'parameter into the XMLDocument object using the LoadXml method.

        Dim myXML As New Xml.XmlDocument()
        myXML.LoadXml(myXMLreturnString)

        'Create a variable to reference the list of NVLs returned by the
        'function. Each NVL child element in the string returned by the
        'function will be considered a separate node within this list.

        Dim myXmlNodeList As Xml.XmlNodeList
        myXmlNodeList = myXML.SelectNodes("/iLONDataServer/NVL")

        'Create a reference to a user-specified NVL from the list returned
        'by DataServerList.

        Dim myXmlNode As Xml.XmlNode
        Const NV_INDEX As Integer = 0
        myXmlNode = myXmlNodeList.Item(NV_INDEX)

        'Insert the <iLONDataServer> element around the node to match the
        'format required within the <Data> parameter of the DataServerGet
        'function. Call the DataServerGet function, using the created
        'string as input. The function will return the configuration of
        'the data point with index number NV_INDEX.

        myXMLreturnString =
            myWebReference.DataServerGet("<iLONDataServer>" &_
            myXmlNode.OuterXml & "</iLONDataServer>")

'Store the <Result> parameter returned by DataServerGet in the
'XMLDocument object created earlier. Then, reference the NVL 'element
included in the return string in the myXmlNode variable.

        myXML.LoadXml(myXMLreturnString)
        myXmlNodeList = myXML.SelectNodes("/iLONDataServer/NVL")
        myXmlNode = myXmlNodeList.Item(NV_INDEX)

        'Insert updated text into the <UCPTlocation> property of the
        'string returned by DataServerGet. Then pass the string to
        'DataServerSet. The <UCPTlocation> of the data point will be
        'updated in the Data Server when DataServerSet is called. The
        '<UCPTlocation> property is the third property within each <NVL>
        'element. As a result, the index value fed to the ChildNodes
        'property must be 2, since the index values are 0-based.

        Dim myNewLocation As String
        myNewLocation = "\Building 1\Room 3\Ceiling "
        Const UCPTlocation As Integer = 2
        myXmlNode.ChildNodes(UCPTlocation).InnerText = myNewLocation
        myXMLreturnString = myWebReference.DataServerSet(myXML.OuterXml)


        'Verify that the index number returned by DataServerSet matches
        'the index number supplied in the input to the function. If they
        'match, the data point was updated successfully.
        myXML.LoadXml(myXMLreturnString)
        myXmlNodeList = myXML.SelectNodes("/iLONDataServer/NVL")
        myXmlNode = myXmlNodeList.Item(NV_INDEX)
        Const UCPTindex As Integer = 0
        If myXmlNode.ChildNodes(UCPTindex).InnerText <> NV_INDEX _
             Then MsgBox("FAILED")
        End If
```

14.2.2.2 Reading Data Logs

The following programming sample invokes the DataLoggerRead function, using a static XML string as input. It then parses the string returned by DataLoggerRead using an XMLDocument object, and writes the information it extracts to a database using SQL. This code includes comments explaining each step of this process.

```
'Create an instance of the i.LON100's WSDL file.
Dim ilon As New WebReference1.iLON100()

'Specify the IP location of the i.LON 100 by replacing the default
'"localhost" with the IP address used by the i.LON 100.
s = ilon.Url
ilon.Url = s.Replace("localhost", "10.5.250.59")

Dim ret As String
Dim data As String

'Create the XML string to be passed to DataLoggerRead. This program
'will read log entries from the Data Logger using index number 0.
data="<iLONDataLogger><Log><UCPTindex>0</UCPTindex></Log></iLONDataLogger>"

'Call DataLoggerRead and send the SOAP message.
ret = ilon.DataLoggerRead(data)

'Prepare to parse the XML string returned by the function. Create an
'XML document object to store the return string, and an Xpath navigator
'object to navigate through the string and return the properties you
'want.
Dim xmlDoc As New System.Xml.XmlDocument()
Dim xmlNav As System.Xml.XPath.XPathNavigator

'Parse the XML string returned by DataLoggerRead.
 xmlDoc.LoadXml(ret)
 xmlNav = xmlDoc.CreateNavigator()

'Create iterators to store the values you want to extract from the
'return string.

Dim dataValues As System.Xml.XPath.XPathNodeIterator
Dim timestamps As System.Xml.XPath.XPathNodeIterator
Dim pointNames As System.Xml.XPath.XPathNodeIterator
Dim locations As System.Xml.XPath.XPathNodeIterator
Dim source As System.Xml.XPath.XPathNodeIterator
Dim units As System.Xml.XPath.XPathNodeIterator
Dim statuses As System.Xml.XPath.XpathNodeIterator
dataValues = xmlNav.Select("/iLONDataLogger/Log/Element/UCPTvalue")
timestamps = xmlNav.Select("/iLONDataLogger/Log/Element/UCPTlogTime")
pointNames = xmlNav.Select("/iLONDataLogger/Log/Element/UCPTpointName")
locations = xmlNav.Select("/iLONDataLogger/Log/Element/UCPTlocation")
address=xmlNav.Select("/iLONDataLogger/Log/Element/UCPTlogSourceAddress")
units = xmlNav.Select("/iLONDataLogger/Log/Element/UCPTunit")
statuses = xmlNav.Select("/iLONDataLogger/Log/Element/UCPTpointStatus")


'Process through the XML string returned by the function, and add each
'property to the SQL data table.
While (dataValues.MoveNext())
      timestamps.MoveNext()
      pointNames.MoveNext()
      locations.MoveNext()
      address.MoveNext()
      units.MoveNext()
      statuses.MoveNext()

      s = "insert VACOMLOG (timestamp, unit, pointname, value, _
              location, address, status) "
      s = s & " values('" & timestamps.Current.Value() & "', '"
      s = s & units.Current.Value() & "', '"
      s = s & pointNames.Current.Value() & "', '"
      s = s & dataValues.Current.Value() & "', '"
      s = s & locations.Current.Value() & "', '"
      s = s & address.Current.Value() & "', '"
      s = s & statuses.Current.Value() & "')"
      sqlCmd.CommandText = s
      sqlCmd.ExecuteNonQuery()

      printf(dataValues.Current.Value() & "  " &    _
          timestamps.Current.Value())
  End While
```

14.2.3 Using DataSets

The following subroutine uses DataSets to construct the input to be supplied to the DataServerRead function. Using DataSets to construct your XML strings provides several advantages.

It allows you to assign each property defined in the input string a variable type. This may be useful if you want to restrict the values a user can assign for a certain property. For example, by storing the value to be assigned a property like <UCPTminDeltaTime> in an Integer variable, you could limit the values that could be assigned to that property to numbers. In Visual Basic .NET, types such as UINT16 can further help restrict the user by enforcing user defined minimum and maximums.

Similarly, by storing the string to be assigned to the <UCPTdescription> on a String variable, you could limit the length of that description. Generally, using types reduces the chance of errors occurring in your SOAP message. For example, a user could not attempt to pass in a string containing letters as the value for an index number, or any other property that requires an integer as its value type. In addition, performing error checking in your application will provide responsive error handling.

The DataSet is a relational structure that can be easily traversed and manipulated when you parse the XML string retrieved by the function. You can use the DataSet as a data source for many OLE and ActiveX controls. In addition, you can use the DataSet with .NET Data Provides to write the returned values to a database of your choice.

14.2.3.1 DataSets and XML Schemas

The data types and relational structure of a DataSet are defined by an XML schema. You must add a schema to your project before working with DataSets. To add a schema to your project:

1) Right-click on the Project name in the Solution Explorer and select **Add>Add New Item** from the pop-up menu. This opens the Add New Item – Solution Options window.

2) Select **DataSet** and click **Open**. Then, click the XML tab at the bottom of the window and enter the schema for your project into the text box.

3) You can generate the schema from the sample <Data> parameters documented in this manual by using an XSD Schema Generator. Alternatively, you could review the XML Schema guidelines to construct a schema from scratch:

http://www.w3.org/XML/Schema

The following represents the schema you might generate for the XML string to be included in the <Data> parameter to be supplied to the DataServerList function. This schema could be used with the code sample included later in this section. You need to specify two schemas when using DataSets, one for the <Data> parameter you will supply to the function, and one for the <Result> parameter that will be retrieved by the function.

The following represents an XML schema you could use for the <Data> parameter:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="DataServerList_Data_Element" elementFormDefault="qualified"
attributeFormDefault="qualified" xmlns="http://tempuri.org/DataServerList_Data.xsd"
xmlns:mstns="http://tempuri.org/DataServerList_Data.e"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-
msdata">
   <xs:element name="DataServerList_Data_Element" msdata:IsDataSet="true">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="iLONDataServer">
               <xs:complexType>
                  <xs:choice maxOccurs="1">
                     <xs:element name="UCPTdataPointType" type="xs:string" minOccurs="0"/>
                     <xs:element name="UCPTsetting" type="xs:string" minOccurs="0" />
                     <xs:element name="UCPTstartIndex" type="xs:int" minOccurs="0" />
                     <xs:element name="UCPTcount" type="xs:int" minOccurs="0" />
                  </xs:choice>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

The following represents an XML schema you could use for the <Result> parameter:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="DataServerList_Result_Element"
targetNamespace="http://tempuri.org/DataServerList_Data.xsd" elementFormDefault="qualified"
attributeFormDefault="qualified" xmlns="http://tempuri.org/DataServerList_Data.xsd"
xmlns:mstns="http://tempuri.org/DataServerList_Data.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-
msdata">
   <xs:element name="DataServerList_Result_Element" msdata:IsDataSet="true">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="iLONDataServer">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="DpNVL" minOccurs="0" maxOccurs="1">
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="SCPTobjMajVer" type="xs:string" minOccurs="0" />
                              <xs:element name="SCPTobjMinVer" type="xs:string" minOccurs="0" />
                              <xs:element name="UCPTlastUpdate" type="xs:string" minOccurs="0" />
                              <xs:element name="UCPTlifeTime" type="xs:string" minOccurs="0" />
                           </xs:sequence>
                        </xs:complexType>
                     </xs:element>
                     <xs:element name="NVL" minOccurs="0" maxOccurs="unbounded">
                       <xs:complexType>
                           <xs:sequence>
                             <xs:element name="UCPTindex" type="xs:string" minOccurs="0" />
                             <xs:element name="UCPTpointName" type="xs:string" minOccurs="0" />
                             <xs:element name="UCPTlocation" type="xs:string" minOccurs="0" />
                           </xs:sequence>
                        </xs:complexType>
                     </xs:element>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

14.2.3.2 Wrapping Elements

Once you have added the XML schema to your project, you can begin writing code to use the schema, and generate the input XML string to be passed to the <Data> parameter of your function.

However, you should note that a DataSet contains two wrapping elements. The root element reperesents the DataSet. The children of the root element represent tables that will contain the data to be included in your input XML string.

The following is an example of an XML String generated by a DataSet:

<A><B><UCPTdataPointType>NVL</UCPTdataPointType>**</B></A>**

A, the parent element, represents the DataSet Name. **B** represents the Table Name. This must match the name of the child element below the <Data> element in the example SOAP messages shown in Chapters 5-13 of this document. In this case, this must be iLONDataServer, since the input string being generated by the DataSet will be passed to the DataServerRead function.

Before making a SOAP call such as DataServerList using the above string, the parent element **A** included in the DataSet must be removed, and the above message must be reduced to:

<**B**><UCPTdataPointType>NVL</UCPTdataPointType></**B**>

As a result, the SOAP message generated by the application will include only a single parent element: Data. The subroutine included in the following section, *DataSets Programming Sample*, removes the parent element from the string generated by the DataSet.

14.2.4 DataSets Programming Sample

The following SubRoutine utilizes ADO.NET DataSets to construct XML strings to be passed to the DataServerRead function, and to retrieve data from the XML strings returned by the function. The project containing this code must contain XML schemas for the <Data> parameter you will supply to the function, and for the <Result> parameter the function will return, as described in the previous section.

```
'Add an instance of  the i.LON 100 WSDL file to the project.
'ILON100_Reference is the name assigned to the instance of the
'WSDL file in the Solution Explorer.
Dim myWebReference As New iLON100_Reference.iLON100()

'Create a DataSet defined by the XML Schema that has been added
'to this project. Note the use of the schema ID
'(DataServerList_Data_Element) here.
Dim myDataSet_Data As New DataServerList_Data_Element()

'Create a DataTable for the schema. This data added to this table
'will be used to create the XML string to be inserted into the
'<Data> parameter when DataServerRead is called, and the SOAP
'message is sent.

  Dim myDataTable_Data As _
     DataServerList_Data_Element.iLONDataServerDataTable
  myDataTable_Data = myDataSet_Data.iLONDataServer

 'A DataTable is comprised of DataRows. When sending a SOAP
 'message, one DataRow is used. When receiving SOAP messages, many
 'DataRows may be used. Rows are used in cases where multiple
 'objects of a type are present.

  Dim myDataRow_Data As _
     DataServerList_Data_Element.iLONDataServerRow
  myDataRow_Data = myDataTable_Data.NewiLONDataServerRow


  With myDataRow_Data
      .UCPTdataPointType = "NVL"
      .UCPTsetting = 0
```

```
    .UCPTstartIndex = 5001
    .UCPTcount = 3
End With

'Add the updated row to the Table
myDataTable_Data.AddiLONDataServerRow(myDataRow_Data)

'Now the root element must be removed since the SOAP message
'provides the <Data> element as the root element. This is
'described in more detail in Wrapping Elements on page 14-8.
Dim myXML As New Xml.XmlDocument()
myXML.LoadXml(myDataSet_Data.GetXml())

Dim myElement As Xml.XmlElement
myElement = myXML.FirstChild

'The XML string is now ready to be passed to the function.
Dim myXMLString As String
myXMLString = myElement.InnerXml

'Specify IP location of the i.LON 100 by replacing the default
' "localhost" with the IP address used by the i.LON 100.
Dim myURL As String
myURL = myWebReference.Url()
myWebReference.Url = myURL.Replace_
       ("localhost", "192.168.1.253")


'Call DataServerList and send the SOAP message.
myXMLString = myWebReference.DataServerList(myXMLString)

'To drop the returned XML string into a DataSet, you must first
'pass it to a reader. The reader will provide the base
'functionality required to read the data in the result DataSet
'to the project.

Dim myReader As System.IO.StringReader
myReader = New System.IO.StringReader(myXMLString)

'Pass the reader to a DataSet type defined by the result
'DataSet created for the project. In this case, this DataSet is
'called DataServerList_Result.xsd.

Dim myDataSet_Result As New DataServerList_Result_Element()
myDataSet_Result.ReadXml(myReader, XmlReadMode.InferSchema)

'The DpNVLDataTable that stores the retrieved data is a table
'defined in the DataServerList_Result_Element schema added to
'the project.

Dim myDataTable_DpNVL As _
    DataServerList_Result_Element.DpNVLDataTable
myDataTable_DpNVL = myDataSet_Result.DpNVL

Dim myDataTable_NVL As _
    DataServerList_Result_Element.NVLDataTable
myDataTable_NVL = myDataSet_Result.NVL
```

```
'Get the first and only DpNVL returned. Although the schema
'created for the project specifies that only 1 element can
'exist in this table, an index is still required to access the
'collection.

Dim myDataRow_DpNVL As DataServerList_Result_Element.DpNVLRow
myDataRow_DpNVL = myDataTable_DpNVL.Rows(0)

Dim myDataRow_NVL As DataServerList_Result_Element.NVLRow

Dim i As Int16
RichTextBox1.Text = ""

'A DataSet can serve as a Data Source for many OLE and ActiveX
'controls. Normally, the following code is not necessary. The
'following loop traverses the collection of NVL points stored
'in the DataSet returned by the function and displays them in a
'text box, line by line.

For i = 0 To myDataTable_NVL.Rows.Count - 1
    myDataRow_NVL = myDataTable_NVL.Rows(i)

    With myDataRow_NVL
        RichTextBox1.Text = RichTextBox1.Text & _
                      .UCPTindex & " " & _
                      .UCPTlocation & " " & _
                      .UCPTpointName & vbCrLf
    End With
Next
```

15 Manually Modifying an XML Configuration File

You can create and manage the XML configuration files of your *i.*LON 100 manually, or with the *i.*LON 100 SOAP interface. This section describes how to create an XML file and download it into the proper directory of the *i.*LON 100, and how to access an XML file that has already been created, modify it, and download it back to the *i.*LON 100.

Echelon strongly recommends that you use the SOAP interface to manage the XML configuration files. The *i.*LON 100 performs error-checking on all data written in a SOAP message, so that invalid data is not written to any of the XML files. The *i.*LON 100 will not perform error-checking on any XML files downloaded to it via FTP, and so manually editing the XML files may cause boot errors.

Additionally, SOAP messages can be sent to the *i.*LON 100 while it is operating, and the XML files affected by the SOAP messages will be updated without requiring a reboot. If you manually edit the XML files using the procedures described in this chapter, you will need to reboot the *i.*LON 100.

15.1 Creating an XML File

The following procedure describes how to create an XML file, and add it to the configuration directory of the *i.*LON 100:

1. Create the XML file following the guidelines provided in this manual. The documentation for each application describes the format the XML file must be created with, and describes the properties you must define in each XML file.

Be sure to save the file using the file names provided in this document. You should use a text editor such as Microsoft® XML Notepad to create your XML file.

2. Use an FTP client application to open an FTP session to the *i.*LON 100. You can connect to the *i.*LON 100 by specifying either its IP address or its hostname.

The default user name and password for the *i.*LON 100 is *ilon*.

3. Insert the XML file in the applicable directory. Most of the XML files described in this document belong in the /root/config/software directory. Refer to the documentation of each application in this manual for more information.

4. Close the FTP session and reboot the *i.*LON 100. When the *i.*LON 100 reboots, it will read the new XML files and adjust its configuration accordingly.

15.2 Modifying an XML File

The following procedure describes how to access an XML file that has already been added to the *i.*LON 100, and how to modify it:

1. Use an FTP client application to open an FTP session to the *i.*LON 100. You can connect to the *i.*LON 100 by specifying either its IP address or its hostname.

The default user name and password for the *i.*LON 100 is *ilon*.

2. Access the directory of the XML file you want to modify. Most of the XML files described in this document can be found in the /root/config/software directory of the *i.*LON 100. Refer to the documentation of each application in this manual for more specific information.

3. Open the XML file you want to modify using Microsoft XML Notepad or any other text editor. Modify the XML file as you like, but be sure that the format of the XML file remains intact.

4. Save the XML file and return it to the directory you took it from. Close the FTP session.

5. Reboot the *i.*LON 100. When the *i.*LON 100 reboots, it will read the modified XML file and adjust its configuration accordingly.

15.3 Copying XML Files Between *i.*LON *100s*

You can copy the configuration of one *i.*LON 100 application into another *i.*LON 100 using FTP as well. When moving XML configuration files from one *i.*LON 100 to another, you must ensure that the data point names referenced in the files being copied correspond to the data points on the target *i.*LON 100.

If this is not the case, you must either modify references to data point names to match existing data point names on the target *i.*LON 100, or add data points to the target *i.*LON 100 which have names that match those in the XML files being copied. In some cases this is not a problem. For example, the EventCalendar.XML file does not contain references to any data point names. Therefore, that file can be copied from one *i.*LON 100 to another without modification. The same is true for Type Translator rule configuration files.

However, for most applications you must change the names. For example, in the AlarmGenerator.XML file, you must verify the <UCPTpointName> property for the following elements before copying the XML file to the target *i.*LON 100: <InputDataPoint>, <CompareDataPoint>, <AlarmDataPoint> and <Alarm2DataPoint>.

This procedure describes how to copy an XML file from one *i.*LON 100 to another:

1. Use an FTP client application to open an FTP session to the *i.*LON 100 containing the XML file, or files, you want to copy. You can connect to the *i.*LON 100 by specifying either its IP address or its hostname.

The default user name and password for the *i.*LON 100 is *ilon*.

2. Access the directory of the XML file you want to copy. Most of the XML files described in this document can be found in the /root/config/software directory of the *i.*LON 100. Refer to the documentation of each application in this manual for more specific information.

3. Open the XML file you want to modify using Microsoft XML Notepad or any other text editor. Save it locally, and close the FTP session.

4. Use an FTP client application to open an FTP session to the second *i.*LON 100. The default user name and password for the *i.*LON 100 is *ilon*.

5. Access the directory of the XML file you are going to copy into the *i.*LON 100. Most of the XML files described in this document can be found in the /root/config/software directory of the *i.*LON 100. Refer to the documentation of each application in this manual for more specific information.

6. Insert the XML file saved in step 3 into the appropriate directory of the *i.*LON 100.

7. Reboot the *i.*LON 100. When the *i.*LON 100 reboots, it will read the new XML file and adjust its configuration accordingly.

Index