ECHELON®

2.0

i.LON® SmartServer 2.0 Programmer's Reference

078-0347-01D

Table of Contents

1 Introduction to the SmartServer SOAP/XML Interface

The SmartServer 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 with the SmartServer Web pages, as described in the *i.LON SmartServer 2.0 User's Guide*. The *i.LON SmartServer 2.0 User's Guide* provides instructions to follow when configuring the SmartServer applications with the SmartServer Web interface, as well as general information on the different SmartServer applications, and guidelines to follow when using these applications.

You can also use the SOAP/XML interface provided with the SmartServer to configure 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 a protocol for exchanging XML-based messages over TCP/IP networks, normally using HTTP. SOAP enables different applications and devices to communicate with each other, regardless of platform, by sending SOAP messages to each other.

The configuration of each SmartServer application is stored in an XML file. The SmartServer 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 SmartServer 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 SmartServer. The content of the SOAP message is based on the input you supply to the function. The SmartServer 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 SmartServer is operating.

It is important to note that the XML files described in this document store the configurations of the SmartServer applications. They do not store the data generated by these applications. The real-time data generated by the SmartServer's applications is stored in RAM and log data are stored on the flash disk.

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

1.1 About This Document

This document describes the XML files that store the configurations of the various SmartServer applications, and the SOAP functions you can use with each application. The SOAP interface provided with the SmartServer 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 SmartServer applications by manually creating and modifying the XML configuration files. Once you have created the XML files, you can download them to the SmartServer via FTP. The SmartServer 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 SmartServer, and instructions to follow when downloading these files to the SmartServer. The XML files used by the SmartServer 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 SmartServer. The SmartServer 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 SmartServer 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 SmartServer while it is operating, and the SmartServer 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 SmartServers, and would like to use the same configuration on each one. In that case, you could configure one of the SmartServers, copy its XML files, and download them to the appropriate directories of the other SmartServers to obtain the same configuration in all of them.

1.2 Programming Samples

Chapter 21 of this document includes programming samples written in Visual C# .NET, Visual Basic .NET, and Java 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 Chapters 2 and 3 before proceeding to the rest of this document and learning about the functions and applications of the SOAP/XML interface. Chapter 2, *SOAP Messages and the SmartServer WSDL*, describes the WSDL file which defines the SOAP/XML interface. Chapter 3, *SmartServer Applications and the SOAP/XML Interface*, provides an overview of the SmartServer applications and includes a roadmap to follow when configuring those applications with a SOAP application.

You can begin to learn how to program the SmartServer using SOAP/XML using the iLON SOAP Tester (version 2.0.3994). The SOAP Tester is an unsupported engineering-level tool provided by Echelon that lets you perform functional testing of the SmartServer's pre-defined SOAP functions and your user-defined SOAP functions. You can download the SOAP Tester from the i.LON SmartServer Community Web site at *http://ilonsmartserver.com/files*. If you have a SmartServer 2.0 (Release 4.03), the SOAP Tester is also included on the *i*.LON SmartServer 2.0 DVD in the **iLon100\iLon100\unsupported\SoapTester** folder. For more information on using the SOAP Tester, see *Appendix A: SOAP Tester Example*.

You can also learn how to program the SmartServer using SOAP/XML by installing an HTTP debugging proxy program like Charles on your computer. The Charles proxy records all SOAP and other HTTP traffic to and from your computer. You can use Internet Explorer to perform normal operations with the SmartServer Web interface and record the SOAP messages that are being sent to the SmartServer. You can download a free trial version of the Charles proxy from the Charles Web site. For more information on using and downloading the Charles proxy, go to the Charles Web site at *www.charlesproxy.com*.

1.4 SmartServer SOAP/XML Interface Upgrades

The format and contents of the SOAP messages you can send to a SmartServer are defined by the SOAP namespace that the SmartServer is using. The SOAP/XML interface used for the SmartServer (software version 4.0) is new. It uses a common set of generic methods (list, get, set, delete, read, write, clear, and invoke) for retrieving and configuring the data of the various SmartServer applications. This differs from the SOAP/XML interface that was used for *e3* (software version 3.0) release, in which each application had its own specialized set of messages and structures.

There have been three different SOAP namespaces introduced during the four releases of the *i*.LON servers. The version 1.0 namespace was introduced for Release 1.0, the version 1.1 namespace was

introduced for the *e2* release, and the version 3.0 namespace was introduced for the *e3* release. Support for the different namespaces as follows:

- *i*.LON 100 servers running the Release 1.0 software only support the version 1.0 namespace. This means that a SmartServer running the Release 1.0 software can only respond to SOAP messages from other SmartServers running the Release 1.0 software.
- *i*.LON 100 servers running the *e2* software support the version 1.0 and 1.1 namespaces. This means that an *i*.LON 100 server running the *e2* software can respond to SOAP messages from other *i*.LON 100 servers that are running the Release 1.0 or the *e2* software.
- *i*.LON 100 servers running the *e3* software support the version 1.1 and 3.0 namespaces. This means that *i*.LON 100 servers running the *e3* software can respond to SOAP messages sent from other *i*.LON 100 servers that are running the *e2* or *e3* software.
- SmartServers running the SmartServer software support only the version 4.0 namespace. This means that SmartServers running the SmartServer software can respond to SOAP messages sent from other SmartServers. SmartServers running the SmartServer software cannot respond to SOAP messages sent from *i*.LON 100 servers running the *e3* software.

The following section, *Version 4.0 SOAP Message Name Schema*, describes the changes made to the SOAP/XML interface between SOAP namespace versions '3.0' and '4.0' (i.e. between the *e3* and SmartServer releases).

Note: This manual often refers to the *i*.LON 100 and SmartServer releases by the version numbers used in their SOAP namespace. For example, the SmartServer release is referred to by its software version number, which is '4.0', and the *i*.LON 100 *e3* release is referred to as '3.0'. For more information on the SOAP namespace, see *Chapter 2* of this document.

1.4.1 Version 4.0 SOAP Message Name Schema

The SOAP/XML interface used for the SmartServer (version 4.0) uses a new message name schema. It uses a common set of generic methods (*List, Get, Set, Delete, Read, Write, Clear*, and *Invoke*) for retrieving and configuring the data of the various SmartServer applications. This section provides an overview of the changes made to the SOAP message name schema from version 3.0.

1.4.1.1 Version 3.0 Message Name Schema

The message name schema used for version 3.0 was *<Application>_<Message>_<Parameter>*. The *Parameter* was optional and normally not used. The *Message* was normally *List*, *Get*, *Set* or *Delete*. The following provides examples of version 3.0 request messages:

TypeTranslator_List, TypeTranslator_Get_Rule, Read, DriverMOD_Set_Template

The response messages were identified by the message name plus the string *<Response>*.
The schema was: *<Application>_<Message>_<Parameter>Response*.

With the version 3.0 message name schema, there were numerous specialized SOAP messages and SOAP structures, resulting in limited flexibility and intricate handling of a large number of Java or C# proxy classes.

1.4.1.2 Version 4.0 Message Name Schema

Version 4.0 uses a single set of messages that are common to all applications. The messages for retrieving and modifying configuration data consists of *List, Get, Set,* and *Delete*, and the messages for retrieving and changing state information are *Read* and *Write.*

2 SOAP Messages and the SmartServer 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:

- *SmartServer Naming Structure*. The section describes how the SmartServer's data configuration is organized.
- *SmartServer WSDL File*. This section describes the version 4.0 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*. This section describes the security provided by the SmartServer for SOAP applications.
- *SOAP Request/Response Structure*. This section demonstrates the structure of the SOAP request and response messages used by Version 4.0.
- *SOAP Message Formats*. This section describes the formats that must be used for all SOAP messages that are sent to and from the SmartServer. A SOAP message is sent to the SmartServer each time you invoke any of the functions described in this document.
- *Data Point References*. The section describes the SmartServer's new method for referencing data points on the SmartServer's embedded applications.
- *UCPTCurrentConfig*. This section describes the new <UCPTcurrentConfig> element included in *List* response messages. This element indicates the namespace version of the client that last set the configuration of an item.
- *Fault Structure*. This section describes the new <Fault> structure, which combines the *faultcode* and *faultstring* elements and enables a client to check only for the existence of a fault structure instead of having to check for both elements.
- *LonString Type*. This section describes the *E_LonString* type that is used for enumerations or element values that have format descriptions that depend on references to other data point formats.
- *SOAP Examples*. This section includes SOAP examples that demonstrate how to use the Version 4.0 SOAP interface to get the configuration of data point, write a value to a data point in a Web connection, and read the data in a data logger.

2.1 SmartServer Naming Structure

The naming convention used for the SmartServer's configuration and data items follows the LONWORKS network hierarchy (*network*/*channel*/*device*/*functional block*/*data point*).

When the SmartServer accesses a device, that device is attached to a given channel in a network. In the same manner, a data point exists on a functional block that is part of the device. This structure is reflected by the *network*/*channel*/*device*/*functional block*/*data point* naming scheme.

Assuming that there is a channel named "*myChannel*" on a network called "*myNetwork*", the name of a device accessed through "*myChannel*" must begin with "*myChannel*". For example, the default <UCPTname> property of the internal automated system device on the SmartServer (**i.LON App**) is **Net/LON/i.LON App**. This means that the SmartServer is located on a channel named *LON* on a network named *Net*. Further consider that **i.LON App** has a **Digital Input 1** functional block that contains an **nvoClsValue_1** data point. By default, the <UCPTname> property of the **nvoClsValue_1** data point is **Net/LON/iLON App/Digital Input 1/nvoClsValue_1**. Note that a <UCPTname> property has to have at least the size of one char and must be unique in the *network*/*channel*/*device*/*functional block*/*data point* collection.

2.2 SmartServer WSDL File

Each SmartServer includes two WSDL (Web Service Description Language) files: **iLON100.wsdl** and **iLON100_System.wsdl**.

The **iLON100.wsdl** file defines most of the SmartServer SOAP/XML interface, and contains all the information an application will require to use the SOAP/XML interface. The **iLON100_System.wsdl** contains the system service methods used to check and configure the SmartServer's settings.

When writing applications to use the SOAP/XML interface, some tools can import these WSDL files and automatically build a class structure for sending and receiving each message. The WSDL files are compatible with numerous programming development environments, such as Microsoft® Visual Studio® 2008, Microsoft Visual Studio 2005, and Eclipse JAVA EE.

For more detailed information on using a WSDL file as a web service in a .NET programming environment, see Chapter 20. In addition, Chapter 20 contains step-by-step instructions you can follow when you reference the version 4.0 WSDL file with a Microsoft Visual Studio project. For more detailed information on using a WSDL file as a web service in a Java programming environment, see Chapter 22.

2.3 Security

You can add a basic level of security to the SOAP/XML interface with the **i.LON Web Server Security and Parameters** program. You can use this utility to add password protection to all web content served by the SmartServer. Basic Access Authentication is the security mechanism used by the SmartServer 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 SmartServer to be authenticated, use the i.LON Web Server Security and Parameters program to password protect the SmartServer SOAP endpoint at the following path: /WSDL/v4.0/iLON100.WSDL.

A user name and password will then be required each time a SOAP message is sent to the SmartServer. 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 user name and password to authenticate access to Web pages that send or receive multiple SOAP messages. If a SOAP message is sent to a SmartServer 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 Appendix C of the *i.LON SmartServer 2.0 User's Guide.*

To protect FTP access to the XML configuration files, the SmartServer requires a user name and password for every FTP session. This username and password default to "ilon", and can be re-defined with the **Setup - Security** Web Page. See Chapter 3 of the *i.LON SmartServer 2.0 User's Guide* for how to use this Web page.

2.4 SOAP Request and Response Message Structure

This section demonstrates the generic format of a complete request/response transaction. Italicized text denotes type definitions and values that are based on the message or use-case. For examples of actual request/response transactions, see Section 2.10, *SOAP Message Examples*.

2.4.1 SOAP Request

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
    <p:messageProperties xmlns:p="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    </p:messageProperties>
 </soap:Header>
 <soap:Body>
    <MessageName xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
      <iLonItem>
         <xSelect>xSelectStatement</xSelect>
         <Item xsi:type="type">
            <UCPTname>networkName/channelName/deviceName/functionBlockName/pointName<UCPTname>
            <Parameter1>Parameter1Value</Parameter1>
            <Parameter2>Parameter2Value</Parameter2>
             ...
         </Item>
         <Item xsi:type="type">
            <UCPTname>networkName/channelName/deviceName/functionBlockName/pointName1<UCPTname>
            <Parameter1>Parameter1Value</Parameter1>
            <Parameter2>Parameter2Value</Parameter2>
             ...
         </Item>
         ...
      </iLonItem>
    </MessageName>
  </soap:Body>
</soap:Envelope>
```

2.4.2 SOAP Response

```
<?xml version="1.0" encoding="utf-8" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
      xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Header>
    <p:messageProperties xmlns:p="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
      <p:UCPTtimeStamp>2005-02-02T11:30:15.220+01:00</p:UCPTtimeStamp>
      <p:UCPTuniqueId>030000066f02</p:UCPTuniqueId>
      <p:UCPTipAddress>172.25.143.222</p:UCPTipAddress>
      <p:UCPTport>80</p:UCPTport>
      <p:UCPTlastUpdate>2005-02-02T11:31:41Z</p:UCPTlastUpdate>
      <p:UCPTprocessingTime>90</p:UCPTprocessingTime>
    </p:messageProperties>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <MessageNameResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
      <iLonItem>
         <UCPTfaultCount>0</UCPTfaultCount>
         <Item xsi:type="type">
            <UCPTname>networkName/channelName/deviceName/functionBlockName/pointName<UCPTname>
            <Parameter1>Parameter1Value</Parameter1>
            <Parameter2>Parameter2Value</Parameter2>
             ...
         </Item>
         <Item xsi:type="type" >
            <UCPTname>networkName/channelName/deviceName/functionBlockName/pointName1<UCPTname>
            <Parameter1>Parameter1Value</Parameter1>
            <Parameter2>Parameter2Value</Parameter2>
             ...
         </Item>
         ...
      </iLonItem>
    </ MessageNameResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.5 SOAP Messages Formats

A SOAP message is sent to the SmartServer each time you invoke a SOAP function. The content of the SOAP message and the affect it has on the SmartServer is based on the input you supply to the function. Using the *Set* function for example, the SmartServer 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 item. The following sections go through each part of the SOAP messages to describe the interface.

SOAP messages are XML documents that are transferred from one entity to another; therefore, the first line in any SOAP message is always the XML version header. SOAP 1.1 and 1.2 both conform to XML 1.0, so this line will not need to change if the SOAP version is updated.

For more information on the types referenced in this section, see the Version 4.0 XML schema type (**iLON100.xsd**), which is installed in the LONWORKS\iLon100\images\iLon100 4.0x\web\WSDL\v4.0 folder on your computer when you install the SmartServer software.

Note: All SOAP messages sent to and from the SmartServer 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 message, as shown in the following sections. However, this restriction is not enforced by the SmartServer software. As a result, the SmartServer 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 SmartServer. 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*.

2.5.1 SOAP Envelope

The SOAP envelope is the highest level in a SOAP message. The SOAP envelope is the highest level of a SOAP message. The SOAP envelope for any message sent to the SmartServer 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 in section 2.3 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>
```

Note: The fourth line of this example includes the symbol "..." This denotes the location of the SOAP body, which is described in section 2.5.3.

2.5.1.1 W3C Namespaces Supported in Version 4.0

Version 4.0 supports the following W3C namespaces:

```
<SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   ...
</SOAP-ENV:Envelope>
```

2.5.2 SOAP Header

The SOAP header contains general information about the entire message. This section is also tightly controlled by the W3C standards. Each 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.

Note: The Version 4.0 SOAP header is only used in a response message (except for the *UCPTvalueFormat* for WebBinder response messages). You do not need to supply this information for a SOAP request.

The following provides an example of a SOAP header in a SOAP response message:

```
<SOAP-ENV:Header>
 <p:messageProperties
  xmlns:p="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <p:UCPTtimeStamp>2008-02-25T15:10:35.360-08:00</p:UCPTtimeStamp>
  <p:UCPTuniqueId>030000197B82</p:UCPTuniqueId>
  <p:UCPTipAddress>10.2.124.82</p:UCPTipAddress>
  <p:UCPTport>80</p:UCPTport>
  <p:UCPTlastUpdate>2008-02-25T19:11:19Z</p:UCPTlastUpdate>
  <p:UCPTprocessingTime>90</p:UCPTprocessingTime>
 </p:messageProperties>
</SOAP-ENV:Header>
```

The SOAP header consists of a complex type with six fields describing different properties of the message:

`<UCPTtimeStamp>`	A time stamp indicating when the message was sent. Per the ISO 8601 standard, the timestamp is in local time, with appended time zone indicators to denote the difference between local time and UTC.
`<UCPTuniqueId>`	A unique identifier assigned to the SmartServer. By default, this is set to the third Neuron ID in the block of 16 Neuron IDs the SmartServer. You can define the unique ID.
`<UCPTipAddress>`	The IP address of the SmartServer that sent the SOAP message. The IP address in this field depends on the network interface used for the SOAP request/response transaction. The SmartServer has two network interfaces: the LAN interface and the PPP interface (modem). As a result, when the SmartServer responds to a SOAP request from the LAN interface, the response header will contain the LAN IP address, and when an outgoing WebBinder SOAP message uses the LAN interface, the SOAP request header will contain the LAN IP address. Conversely, when a SOAP request is received over a PPP connection, including GPRS connections, the SOAP response header will contain the IP address of the PPP interface, and outgoing WebBinder SOAP messages sent over the PPP interface will contain the IP address of the PPP interface in the request header.
`<UCPTport>`	The HTTP port of the SmartServer that sent the SOAP message
`<UCPTlastUpdate>`	A timestamp indicating the last time the configuration of any of the applications of the SmartServer was modified. After a reboot, the timestamp is set to match the reboot time. The timestamp is in local time.
`<UCPTprocessingTime>`	The time elapsed in milliseconds from the server (SmartServer or LNS Proxy Web service) receiving a SOAP request to sending the SOAP response.

2.5.3 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.

Request messages are identified by the message name for the SOAP call, and the response messages are identified by the message name plus the string "*Response*". The message name, which serves as the element name within the XML structure, is uniquely identified by the SmartServer namespace.

Each and every SOAP message in the version 4.0 WSDL has one parameter named <iLonItem>. The <iLonItem> property is a structure derived from *E_xSelect* and has a type of *Item_Coll*. The SOAP body of the sample input message shown in section 2.4 includes the following:

```
 <soap:Body>
    <MessageName xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
      <iLonItem>
         <xSelect>xSelectStatement</xSelect>
         <Item xsi:type="type">
            <UCPTname>networkName/channelName/deviceName/functionBlockName/pointName<UCPTname>
            <Parameter1>Parameter1Value</Parameter1>
            <Parameter2>Parameter2Value</Parameter2>
             ...
         </Item>
         ...
      </iLonItem>
    </MessageName>
  </soap:Body>

  <SOAP-ENV:Body>
    <MessageNameResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
      <iLonItem>
         <UCPTfaultCount>0</UCPTfaultCount>
         <Item xsi:type="type">
            <UCPTindex>0</UCPTindex>
            <UCPTname>networkName/channelName/deviceName/functionBlockName/pointName<UCPTname>
            <Parameter1>Parameter1Value</Parameter1>
            <Parameter2>Parameter2Value</Parameter2>
             ...
         </Item>
         ...
      </iLonItem>
    </MessageNameResponse>
  </SOAP-ENV:Body>
```

2.5.3.1 Fault Messages (Application Layer)

The SOAP body in the response for every function in the SOAP/XML interface contains information indicating whether any errors occurred while processing the request. Each item instance can contain a fault object. An item instance without a fault object indicates an item that was successfully processed.

The following examples demonstrate instances that succeeded (no fault objects), an instance that was applied but is not valid (*faultType="_warning"*), and an instance that was rejected (*faultType="_error"*). The <UCPTfaultCount> tag informs the client about the number of warnings and errors that have occurred. If <UCPTfaultCount> is 0, then no faults occurred.

Example 1 (warning occurs at the item level):

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>1</UCPTfaultCount>
    <Item>
        <fault>
            <faultcode faultType="_warning">4</faultcode>
            <faultstring xml:lang="en-US">fault string</faultstring>
        </fault>
        <UCPTname>networkName/channelName/deviceName/myAG</UCPTname>
    </Item>
    <Item>
        <UCPTname>networkName/channelName/deviceName/yourAG</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

Although an item instance has a warning fault code, it can still be processed. All other item instances with a fault object can be processed.

Example 2 (error occurs at the item level):

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>1</UCPTfaultCount>
    <Item>
```

```
        <fault>
            <faultcode faultType="_error">6</faultcode>
            <faultstring xml:lang="en-US">Instance doesn't exist</faultstring>
        </fault>
        <UCPTname>Net/VirtCh/iLON App</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

Example 3 (error occurs at the global level):

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <UCPTfaultCount>1</UCPTfaultCount>
            <fault>
              <faultcode faultType="_error">3</faultcode>
              <faultstring xml:lang="en-US">invalid xSelect expression</faultstring>
            </fault>
    </iLonItem>
</SetResponse>
```

Because the error occurs at the global level (the error does not occur with a specific item instance), none of the items are processed.

2.5.3.2 Fault Messages (SOAP Layer)

The SOAP fault message is a standard way to report back unexpected SOAP behavior to the originator of the message. This response message has a different format than the response message for a message call that succeeds, and it adheres to a standard format for SOAP 1.1 fault messages. In Version 4.0, all applications will stop processing when an error occurs. In addition, the applications will return with the following message:

```
<?xml version="1.0" encoding="utf8" ?>
<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>1</SOAP-ENV:faultcode>
            <SOAP-ENV:faultstring>soap fault</SOAP-ENV:faultstring>
            <SOAP-ENV:detail>Details</<SOAP-ENV:detail>>
        </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.5.3.3 Error Codes

The error codes used by the SmartServer applications are as follows:

Error Code	Error Description	Comment
0	No Error	
1	Unknown message call	
2	Parameter error	
3	XML/Parser Error	
4	Tag missing	
5	<UCPT name> missing	
6	<UCPT name> not found	
7	<UCPT name> invalid	
8	Can’t create	For example FB, Data point
9	Can’t delete	For example FB, Data point
10	Can’t set	For example FB, Data point
11	Format Error	
12	Command failed	
13	Given Data point has not the given Index	
14	Data point Name not found	

Error Code	Error Description	Comment
15	No Data	

2.5.4 Namespace

The namespace uniquely identifies message names, parameters, and types within the message call. On the SmartServer, the namespace identifier displays both the product name and the embedded software version. This enables a mechanism to distinguish the SOAP interface of different Echelon products and versions. To ensure that this string points to a unique name on the Internet, the namespace includes the echelon.com domain suffix. For release 4.0 of the SmartServer the Namespace is as follows: *http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/*

2.5.5 SOAP Message Schema

Version 4.0 uses a single set of messages that are common to all applications. The message for obtaining a list of items with a specific xSelect type is *List*. The messages for retrieving and modifying configuration data consist of *List*, *Get*, *Set*, and *Delete*. The messages for retrieving and changing dynamic data (for example, data point state and values) are *Read* and *Write.*

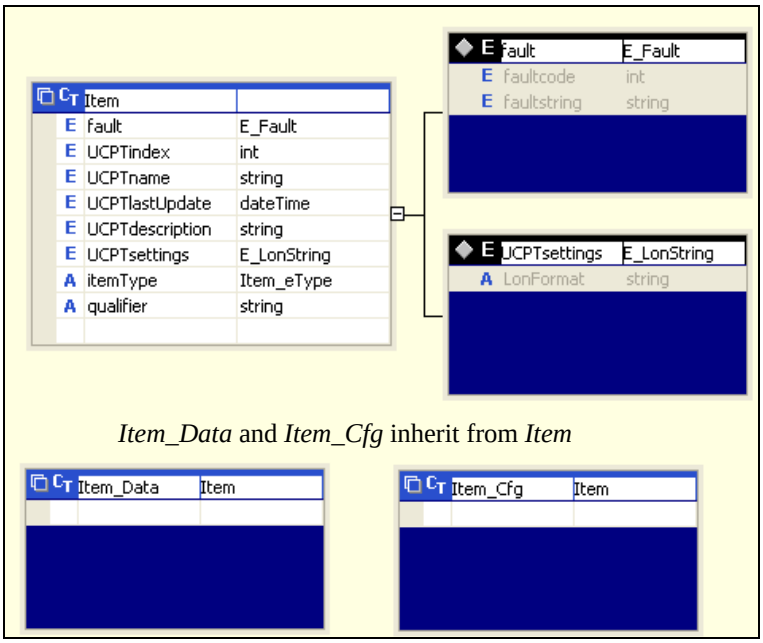
The *Get*, *Set*, and *Delete* messages and the *Read* and *Write* messages contain and return one element of a collection type. There are only three Collection types which contain objects of one of the following types: *Item* (for some request/responses), *Item_Cfg* (for configuration information), *and Item_Data* (for state information).

All types defined in the schema part of the version 4.0 iLon100.wsdl file follow a strict naming convention. The most important naming conventions you need to adhere to are as follows:

- The prefix used is always the destination separated be an "*_*", *e.g.:* (for example, *UFPTalarmNotifier_* or *Dp_*)
- Configuration item types always inherit from *Item_Cfg* and have the postfix "*_Cfg"* (for example, *UFPTalarmNotifier_Cfg* or *Dp_Cfg*).
- State item types always inherit from *Item_Data* and have the postfix "*_Data"* (for example, *UFPTalarmNotifier_Data* or *Dp_Data*).
- Specialized item types have the postfix *"_Invoke*" and are used with the general *Invoke* method (for example, *Dp_ResetPrio_Invoke*).

2.5.6 SOAP Function Types

The *Item* type is the common base type for all other top level types which can be added to one of the Item collections. It contains some common elements and attributes and the *fault* structure. The types *Item_Cfg* and *Item_Data* inherit from *Item*; therefore, *Item_Cfg* is the base types for any configuration information type, and *Item_Data* is the base type for any state information type.



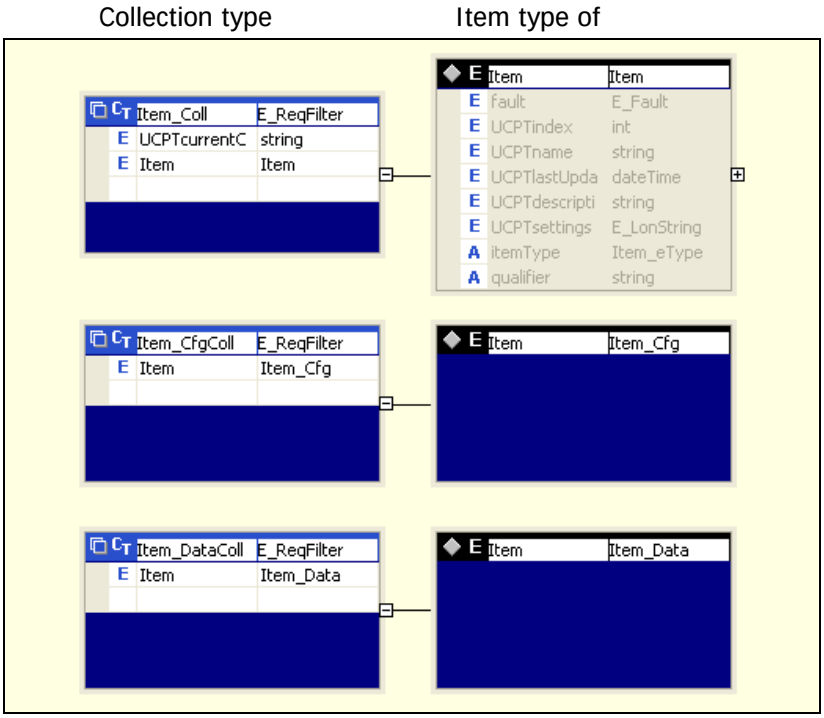
Item_Data and *Item_Cfg* inherit from *Item*

Because *Item* is the base class for all types passed in the *Get* /*Set* /*Delete*, *Read* /*Write*, and *Invoke* functions, you can pass any kind of configuration or state information in the corresponding message.

Message Name	Description	Request Object	Response Object
List	Retrieves a list of *Item*s.	xSelect	Item_Coll
Get	Retrieves the configuration of items.	Item_Coll	Item_CfgColl
Set	Creates items or overwrites the configuration of existing items.	Item_CfgColl	Item_Coll
Delete	Deletes an item.	Item_Coll	Item_Coll
Read	Reads the value and status of items.	Item_Coll	Item_DataColl
Write	Writes values or states to items.	Item_DataColl	Item_Coll
Invoke	Sends a network management command to a LONWORKS device or calls special functions on applications or data points.	Item_Coll	Item_Coll

All requests for information (*Read* and *Get*, and *Delete* and *Invoke* messages) are called with *Item_Coll* as the collection type. *Item_Coll* is a collection of *Item* and can thus contain any type (also *Item_Cfg* and *Item_Data*).

The *Set*-Response and *Get*-Request messages use the *Item_CfgColl*, which is a collection of types containing configuration data. The *Read*-Response and *Write*-Request messages use the *Item_DataColl* collection, which is a collection of types containing dynamic data.



You can cast an item to a more specialized type using meta data (the *xsi:type* attribute) that is passed along with the application data. The *xsi:type* attribute describes the actual type of an item and is passed as an attribute of the <Item> element. If the sent type is the expected base type (*Item*, *Item_Cfg* or, *Item_Data*), no *xsi:type* attribute is sent. For more information on xsi types, go to *www.w3.org/TR/xmlschema-1/#Instance_Document_Construction*.

2.5.7 SOAP Message Attributes

Version 4.0 SOAP request messages support some attributes, and the response messages from the SmartServer always contain all attributes. If the data type of an item is not the declared as a base type in the SOAP request (*Item*, *Item_Cfg*, or *Item_Data*), the *xsi:type* attribute will be added to the item instance. Note that you do not have to set the *xsi:type* attribute in a SOAP request if you are using a SOAP framework like .NET or one of Java's SOAP frameworks. This is because the *xsi:type* attribute is part of the XML Schema specification, and it will be added automatically.

For more information on the *xsi:type* attribute, see the Version 4.0 XML schema type (**iLON100.xsd**), which is installed in the LONWORKS\iLon100\images\iLon100 4.0x\web\WSDL\v4.0 folder on your computer when you install the SmartServer software.

2.5.8 Using xSelect Statements in SOAP Message Requests

You can use *xSelect* statements in *List*, *Get*, *Read*, and *Delete* messages to filter the items returned by the function. The *xSelect* statement queries the item instances in a given data set and returns those items meeting the specified criteria. The *xSelect* statement provides some of the functionality of the xPath language defined by the W3C, except that you can compare strings to dates using compare operators "<", ">", and "=".

Each xSelect statement should specify an xsi (item) type. An xSelect statement may reference an item identifier <UCPTname> and contain some predicates [predicate1] [predicate2]. In any xSelect statement, the predicates may include a <UCPTlastUpdate> expression and a position () expression. Common predicates used include *contains* and *starts-with*.

The predicates are processed as a cascade of queries. For example, you could first use a <UCPTlastUpdate> query to get the items that were updated during a specific time period and then query items 10 to 20 returned by the first result using a position() expression.

The following code samples demonstrate supported xSelect statements.

Example 1 – List or Get all channels on the SmartServer:

```
<iLonItem>
    <xSelect> = "//Item[@xsi:type="Channel_Cfg"] </xSelect>
</iLonItem>
```

Example 2 – List or Get all LONWORKS channels on the SmartServer that were updated after a specific time:

```
<iLonItem>
    <xSelect>//Item[@xsi:type="LON_Channel_Cfg"][UCPTlastUpdate &gt;"2008-04-01T00:00:00"]
    </xSelect>
</iLonItem>
```

Example 3 – List or Get all LONWORKS application devices of a specific type (by name) on a specific channel.

```
<iLonItem>
    <xSelect>//Item[@xsi:type="LON_Device_Cfg"][contains(UCPTname,"Net/LON/DIO")]</xSelect>
</iLonItem>
```

Example 4 – List or Get all instantiated functional blocks on the SmartServer automated systems device [i.LON App (internal)]:

```
<iLonItem>
    <xSelect>//Item[@xsi:type="Fb_Cfg"][starts-with(UCPTname,"Net/LON/")][UCPThidden=0]</xSelect>
</iLonItem>
```

Example 4 – List, Get, or Read all data points on the Digital Input 1 functional block on the internal i.LON App device:

```
<iLonItem>
    <xSelect>//Item[@xsi:type="Dp_Cfg"]
                  [starts-with(UCPTname, "Net/LON/iLON App/Digital Input 1/")]</xSelect>
</iLonItem>
```

Example 5 – Get all SNVT_switch data points on the SmartServer that were updated in the time line defined by <UCPTlastUpdate>:

```
<iLonItem>
    <xSelect>
      //Item[@xsi:type="Dp_Cfg"][UCPTformatDescription="#0000000000000000[0].SNVT_switch"]
            [UCPTlastUpdate &gt;"2008-04-01T00:00:00" and UCPTlastUpdate &lt;"2008-04-07T00:00:00"]
      </xSelect>
</iLonItem>
```

Example 6 – List, Get, or Read the data points on the internal i.LON App device in the time line defined by <UCPTlastUpdate> to return a maximum of the 11th to 29th data points:

```
<iLonItem>
    <xSelect>//Item[@xsi:type="Dp_Cfg"] [starts-with(UCPTname,"Net/LON/iLON App/")]
        [UCPTlastUpdate &gt; "2008-04-01T15:30:21Z" and UCPTlastUpdate &lt; "2008-04-08T15:30:21Z"]
        [position()&gt;10 and position()&lt;30]
    </xSelect>
</iLonItem>
```

Example 7 – List or Get all instantiated Alarm Generators on the internal i.LON App device;

```
<iLonItem>
    <xSelect>//Item[@xsi:type="UFPTalarmGenerator_Cfg"]</xSelect>
</iLonItem>
```

Example 8 – Read the first 10 events scheduled in a Scheduler on the internal i.LON App device:

```
<iLonItem>
    <xSelect>//Item[@xsi:type=”UFPTscheduler_Data”]
           [UCPTname="Net/LON/iLON App/Scheduler"]
           [UCPTeventFilter="EF_SCHEDULE"][position()&lt;10]"
    </xSelect>
</iLonItem>
```

Example 9 – Select a formatter report:

```
xSelect = "//Item[@xsi:type=”TemplateManager_NVT_Cfg”][UCPTlanguage="enu"]
          [UCPTname="#0000000000000000[0].standard"][position()>=0 and
          position()<10]"
```

Notes:

- For *List* functions, you must include an xSelect statement in the SOAP message request.
- For *Get*, *Delete*, and *Read* functions, the xSelect statement in the SOAP message request is optional.
  - If a *Get*, *Delete*, or *Read* function is called only with an xSelect statement, the function returns all items of the specified xsi type that meet the criteria specified in the xSelect statement.
  - If a *Get*, *Delete*, or *Read* function is called with an xSelect statement and one or more item instances, the xSelect statement is overlaid on the item instances. This means that the xSelect statement is an *and* expression (not an *or* expression or an *exclusive or* expression).
- You can only filter for properties that are included in a message response (except for the <UCPTlastUpdate> property, which you can always filter). In a *List* request of Dp_Cfg items (data points on the Data Server) for example, you can filter on the <UCPTname> property, but you cannot filter on the <UCPTformatDescription>. In a *Get* request of Dp_Cfg items, however, you can filter on the <UCPTformatDescription> property because this property is returned in the *Get* message response.
- The SmartServer can only process one item type (xsi:type) per SOAP message (e.g., only Channel_Cfg, LON_Dp_Cfg, or UFPTalarmGenerator). This is also true for xSelect responses (except for UCPTlastUpdate which is a special case), *Set* messages, and other SOAP commands. The first item type specified is always the one returned in the response message.
- Not all properties can be filtered and not all filter combinations work.

2.5.8.1 xsi Types

The following section lists the common xsi (items) types that you will include in xSelect statements. For a list of all the item types included in the SOAP interface, see the Version 4.0 XML schema type (**iLON100.xsd**), which is installed in the LONWORKS\iLon100\images\iLon100 4.0x\web\WSDL\v4.0 folder on your computer when you install the SmartServer software

Driver	Item	xsi type
General (no Driver)	Network	Network_Cfg
	Channel	Channel_Cfg
	Device	Device_Cfg
	Functional Block	Fb_Cfg
	SmartServer Applications	
	Alarm Generator	UFPTalarmGenerator_Cfg
	Alarm Notifier	UFPTalarmNotifier_Cfg
	Analog Functional Block	UFPTanalogFunctionBlock_Cfg
	Calendar	UFPTcalendar_Cfg
	Data Logger	UFPTdataLogger_Cfg
	Digital Input	UFPTdigitalInput_Cfg
	Digital Ouput	UFPTdigitalOutput_Cfg
	Node Object	UFPTnodeObject_Cfg
	Pulse Counter	UFPTpulseCounter_Cfg
	Real-TimeClock	UFPTrealTimeClock_Cfg
	Scheduler	UFPTscheduler_Cfg

Driver	Item	xsi type
	Type Translator	UFPTtypeTranslator_Cfg
	Type Translator Rule	UFPTtypeTranslator_Rule_Cfg
	Data Point (configuration)	Dp_Cfg
	Data Point (data)	Dp_Data
	Data Point Reset Priority	Dp_ResetPrio_Invoke
LONWORKS	Network	LON_Network_Cfg
	Channel	LON_Channel_Cfg
	Device	LON_Device_Cfg
	Router	LON_Device_Router_Cfg
	RNI	LON_Device_RNI_Cfg
	Functional Block	LON_Fb_Cfg
	Network Variable	LON_Dp_Cfg
	Configuration Property	LON_Cp_Dp_Cfg
	Configuration Property File	LON_Cp_File_Cfg
Modbus	Network	MOD_Network_Cfg
	Channel	MOD_Channel_Cfg
	Device	MOD_Device_Cfg
	Functional Block	MOD_Fb_Cfg
	Data Point	MOD_Cp_Dp_Cfg
M_Bus	Network	MBS_Network_Cfg
	Channel	MBS_Channel_Cfg
	Device	MBS_Device_Cfg
	Functional Block	MBS_Fb_Cfg
	Data Point	MBS_Cp_Dp_Cfg
Virtual	Network	Virtual_Network_Cfg
	Channel	Virtual_Channel_Cfg
	Device	Virtual_Device_Cfg
	Functional Block	Virtual_Fb_Cfg
	Data Point	Virtual_Cp_Dp_Cfg

2.6 Data Point References

The SmartServer has a more abstract and flexible method for referencing data points on the SmartServer's embedded applications by their parent functional blocks. Primarily, the data point type is not specified by the tag name (for example, the *SNVT_alarm_2 output* tag is obsolete). In Version 4.0, each functional block representing a SmartServer embedded application has a collection of data point references. The data point type is specified by the *dpType* attribute, and it is a reference to the data point's programmatic name as defined in the functional profile template. Some functional blocks also extend the base data point reference XML type. In these cases, the <DataPoint> tag also has an *xsi:type* attribute with the derived type.

The following code samples demonstrate how data points on the SmartServer's embedded applications are referenced. In the first example, the nvoAgAlarmFlag data point on an Alarm Generator functional block is referenced. In the second example, the nvoDlLevAlarm data point on a Data Logger functional block is referenced:

```
<DataPoint dpType="nvoAlarmFlag" discrim="dir_out">
    <UCPTname>Net/LON/iLON App/Alarm Generator[0]/nvoAgAlarmFlag[0]</UCPTname>
    <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
</DataPoint>
```

```
<DataPoint dpType="nvoLevelAlarm" discrim="dir_out">
    <UCPTname>Net/LON/iLON App/Data Logger[0]/nvoDlLevAlarm[0]</UCPTname>
    <UCPTformatDescription>#0000000000000000[0].SNVT_alarm</UCPTformatDescription>
</DataPoint>
```

2.7 UCPTcurrentConfig

Each application on the SmartServer stores the namespace version of the client that last set its configuration (via a *Set* function) its last configuration. When a client sends a *List* function in a SOAP request, the SOAP response returns the *UCPTcurrentConfig* tag, which contains this information. The client should check this tag because new namespaces may contain new or different tags that older clients will not understand.

If a SmartServer responds with a newer *UCPTcurrentConfig*, the client should allow the user either to continue using the older tool with the risk that it may overwrite changes made with a newer tool, or fail and preserve the configuration created by the new tool.

The following code demonstrates the *UCPTcurrentConfig* tag returned by a SOAP response after receiving a *List* function in a SOAP request.

```
<UCPTcurrentConfig>4.0</UCPTcurrentConfig>
```

2.8 Fault Structure

The *faultcode* and *faultstring* are now combined in a fault structure. If a fault structure exists, both the *faultcode* and *faultstring* elements exist. This enables a client to check only for the existence of a fault structure instead of having to check for both elements.

The following code demonstrates the fault structure used in Version 4.0.

```
<fault>
  <faultcode faultType="_error">4</faultcode>
  <faultstring xml:lang="en-US">fault string</faultstring>
</fault>
```

2.9 LonString type

The Version 4.0 XML schema declares the type *E_LonString*, which is a string type and has a value that represents a LONWORKS value. The *E_LonString* also contains the *LonFormat* attribute which is a string that contains the format description of the item. The WSDL uses the *E_LonString* type for enumerations or element values that have format descriptions that depend on references to other data point formats.

The following code sample demonstrates the *LonFormat* attribute of the *E_LonString* type.

```
<UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch">100.0 1</UCPTvalue>
<UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch.value">100.0</UCPTvalue>
<UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch.state">1</UCPTvalue>
<UCPTvalue LonFormat="UCPTvalueDef">ON</UCPTvalue>
<UCPTpointStatus LonFormat="UCPTpointStatus">AL_NUL</UCPTpointStatus>
```

When you use the *Read* function on a data point on the Data Server (*xsi type* = Dp_Cfg), the *E_LonString* type has a unit attribute that specifies the unit strings defined for the data point.

```
<UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch" Unit="value, state">0.0 0</UCPTvalue>
```

2.10 SOAP Message Examples

The following examples demonstrate how to use the Version 4.0 SOAP interface to get the configuration of data point, write a value to a data point in a Web connection, and read the data in a data logger.

2.10.1 Configuration Data

The following example demonstrates how to use a *Get* function to obtain the configuration of a data point on the Data Server (Dp_Cfg) using the unique <UCPTname> of the data point.

Request Message

```
<?xml version='1.0' encoding='utf-8'?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header>
    <p:messageProperties xmlns:p="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
        <p:UCPTtimeStamp>2008-04-08T11:37:32.156-07:00</p:UCPTtimeStamp>
        <p:UCPTuniqueId>0300000A5BF2</p:UCPTuniqueId>
    <p:UCPTipAddress>10.2.124.53</p:UCPTipAddress>
    <p:UCPTport>80</p:UCPTport>
    <p:UCPTlastUpdate>2008-04-08T18:22:52Z</p:UCPTlastUpdate>
    <p:UCPTprocessingTime>20</p:UCPTprocessingTime>
    </p:messageProperties></SOAP-ENV:Header>
<SOAP-ENV:Body>
    <Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
      <iLonItem>
        <Item xsi:type="Dp_Cfg">
            <UCPTname>Net/LON/iLON App/VirtFb/nviSwitch_3</UCPTname>
        </Item>
      </iLonItem>
    </Get>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response Message

```
<?xml version="1.0" encoding="utf-8" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><SOAP-ENV:Header>
    <p:messageProperties xmlns:p="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
        <p:UCPTtimeStamp>2008-04-08T11:56:41.116-07:00</p:UCPTtimeStamp>
        <p:UCPTuniqueId>0300000A5BF2</p:UCPTuniqueId>
        <p:UCPTipAddress>10.2.124.53</p:UCPTipAddress>
        <p:UCPTport>80</p:UCPTport>
        <p:UCPTlastUpdate>2008-04-08T18:22:52Z</p:UCPTlastUpdate>
        <p:UCPTprocessingTime>46</p:UCPTprocessingTime>
    </p:messageProperties>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
    <GetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
      <iLonItem >
        <UCPTfaultCount>0</UCPTfaultCount>
        <Item xsi:type="Dp_Cfg" >
            <UCPTname>Net/LON/iLON App/VirtFb/nviSwitch_3</UCPTname>
            <UCPTannotation>Dp_In;xsi:type=&amp;quot;LON_Dp_Cfg&amp;quot;</UCPTannotation>
            <UCPThidden>0</UCPThidden>
            <UCPTlastUpdate>2008-03-26T16:16:43.070-07:00</UCPTlastUpdate>
            <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
            <UCPTlength>2</UCPTlength>
            <UCPTdirection LonFormat="UCPTdirection" >DIR_IN</UCPTdirection>
            <UCPTunit field="" >value, state</UCPTunit>
            <UCPTunit field="value" >% of full level</UCPTunit>
            <UCPTunit field="state" >state code</UCPTunit>
            <UCPTbaseType LonFormat="UCPTbaseType" >BT_STRUCT</UCPTbaseType>
            <UCPTmaxFields>2</UCPTmaxFields>
            <SCPTmaxSendTime>0.0</SCPTmaxSendTime>
            <SCPTminSendTime>0.0</SCPTminSendTime>
            <SCPTmaxRcvTime>0.0</SCPTmaxRcvTime>
            <ValueDef>
                <UCPTindex>0</UCPTindex>
                <UCPTname>OFF</UCPTname>
                <UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch" >0.0 0</UCPTvalue>
```

```
            </ValueDef>
            <ValueDef>
                <UCPTindex>1</UCPTindex>
                <UCPTname>ON</UCPTname>
                <UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch" >100.0 1</UCPTvalue>
            </ValueDef>
        </Item>
      </iLonItem>
    </GetResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.10.2 Web Binding

You can use the WebBinder application on the SmartServer to send a SOAP message when a data point update occurs. When the source data point (Dp_Data) in a Web connection is updated, the SmartServer sends a *Write* request message for the transaction, and the response is sent by the receiver, which may be another SmartServer, an LNS server, or a WebBinder Target (a Web server that can process SOAP requests). The following example shows a *Write* message that is sent by a SmartServer and is configured by the WebBinder application.

Request Message

```
<?xml version="1.0" encoding="utf-8" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Header>
    <p:messageProperties xmlns:p="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
        <p:UCPTtimeStamp>2008-04-08T11:14:56.289-07:00</p:UCPTtimeStamp>
        <p:UCPTuniqueId>0300000A5BF2</p:UCPTuniqueId>
        <p:UCPTipAddress>10.2.124.53</p:UCPTipAddress>
        <p:UCPTport>80</p:UCPTport>
        <p:UCPTlastUpdate>2008-04-08T18:10:37Z</p:UCPTlastUpdate>
        <p:UCPTprocessingTime>102</p:UCPTprocessingTime>
    </p:messageProperties>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
    <Write xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
      <iLonItem>
        <UCPTfaultCount>0</UCPTfaultCount>
        <Item xsi:type="Dp_Data">
            <UCPTname>Net/LON/iLON App/Digital Input 1/nvoClsValue_1</UCPTname>
            <UCPTannotation>Dp_Out;xsi:type="LON_Dp_Cfg"</UCPTannotation>
            <UCPThidden>0</UCPThidden>
            <UCPTaliasName>NVL_nvoClsValue_1</UCPTaliasName>
            <UCPTlastUpdate>2008-04-08T11:14:53.599-07:00</UCPTlastUpdate>
            <UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch">0.0 0</UCPTvalue>
            <UCPTvalue LonFormat="UCPTvalueDef">OFF</UCPTvalue>
            <UCPTpointStatus LonFormat="UCPTpointStatus">AL_NO_CONDITION</UCPTpointStatus>
            <UCPTpriority>255</UCPTpriority>
        </Item>
      </iLonItem>
    </Write>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response Message

```
<?xml version="1.0" encoding="utf-8" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Header>
    <p:messageProperties xmlns:p="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
        <p:UCPTtimeStamp>2008-04-08T11:37:32.156-07:00</p:UCPTtimeStamp>
        <p:UCPTuniqueId>0300000A5BF2</p:UCPTuniqueId>
        <p:UCPTipAddress>10.2.124.53</p:UCPTipAddress>
        <p:UCPTport>80</p:UCPTport>
        <p:UCPTlastUpdate>2008-04-08T18:22:52Z</p:UCPTlastUpdate>
```

```
       <p:UCPTprocessingTime>20</p:UCPTprocessingTime>
       </p:messageProperties>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
    <WriteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
      <iLonItem >
        <UCPTfaultCount>0</UCPTfaultCount>
        <Item xsi:type="Dp_Data" >
            <UCPTname>Net/LON/iLON App/Digital Input 1/nvoClsValue_1</UCPTname>
            <UCPTannotation>Dp_Out;xsi:type=&quot;LON_Dp_Cfg&quot;</UCPTannotation>
            <UCPThidden>0</UCPThidden>
            <UCPTaliasName>NVL_nvoClsValue_1</UCPTaliasName>
            <UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch" >100.0 1</UCPTvalue>
            <UCPTvalue LonFormat="UCPTvalueDef" >ON</UCPTvalue>
        </Item>
      </iLonItem>
    </WriteResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.10.3 Data Log Read

You can read the contents of the alarm logs and data logs stored on the SmartSever. The following example demonstrates how to use the *Read* function to read the elements in a Data Log.

Request Message

```
<?xml version="1.0" encoding="utf-8" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Header>
    <p:messageProperties xmlns:p="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
        <p:UCPTtimeStamp>2008-04-08T10:28:36.840-07:00</p:UCPTtimeStamp>
        <p:UCPTuniqueId>0300000A5BF2</p:UCPTuniqueId>
        <p:UCPTipAddress>10.2.124.53</p:UCPTipAddress>
        <p:UCPTport>80</p:UCPTport>
        <p:UCPTlastUpdate>2008-04-07T23:45:19Z</p:UCPTlastUpdate>
        <p:UCPTprocessingTime>54</p:UCPTprocessingTime>
        </p:messageProperties>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
    <Read xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
      <iLonItem>
        <xSelect>//Item [@xsi:type="UFPTdataLogger_Data"]
                        [UCPTlastUpdate &gt; "2008-04-08T00:00:00.000"
                         and UCPTlastUpdate &lt;= "2008-04-08T12:00:00.000"]
                        [position()&lt; 3]
        </xSelect>
            <Item>
                <UCPTname>Net/LON/iLON App/Data Logger[0]</UCPTname>
            </Item>
      </iLonItem>
    </Read>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response Message

```
<?xml version="1.0" encoding="utf-8" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><SOAP-ENV:Header>
    <p:messageProperties xmlns:p="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
        <p:UCPTtimeStamp>2008-04-08T11:30:31.272-07:00</p:UCPTtimeStamp>
        <p:UCPTuniqueId>0300000A5BF2</p:UCPTuniqueId>
        <p:UCPTipAddress>10.2.124.53</p:UCPTipAddress>
        <p:UCPTport>80</p:UCPTport>
        <p:UCPTlastUpdate>2008-04-08T18:22:52Z</p:UCPTlastUpdate>
        <p:UCPTprocessingTime>83</p:UCPTprocessingTime>
```

```
    </p:messageProperties>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
    <ReadResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
      <iLonItem >
        <UCPTfaultCount>0</UCPTfaultCount>
            <Item xsi:type="UFPTdataLogger_Meta_Data" >
                <UCPTname>Net/LON/iLON App/Data Logger[0]</UCPTname>
                <UCPTlastUpdate>2008-04-08T11:30:00.062-07:00</UCPTlastUpdate>
                <UCPTstart>2008-04-08T11:22:42.762-07:00</UCPTstart>
                <UCPTstop>2008-04-08T11:30:00.062-07:00</UCPTstop>
                <UCPTmodificationNumber>0</UCPTmodificationNumber>
                <UCPTlogLevel>8.778</UCPTlogLevel>
                <UCPTtotalCount>59</UCPTtotalCount>
            </Item>
            <Item xsi:type="UFPTdataLogger_Data" >
                <UCPTname>Net/LON/iLON App/Digital Input 1/nvoClsValue_1</UCPTname>
                <UCPTaliasName>NVL_nvoClsValue_1</UCPTaliasName>
                <UCPTlastUpdate>2008-04-08T11:22:42.762-07:00</UCPTlastUpdate>
                <UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch">0.0 0</UCPTvalue>
                <UCPTvalue LonFormat="UCPTvalueDef">OFF</UCPTvalue>
                <UCPTpointStatus LonFormat="UCPTpointStatus">AL_NO_CONDITION</UCPTpointStatus>
                <UCPTpriority>255</UCPTpriority>
                <UCPTmetaDataPath>//*[@xsi:type=“UFPTdataLogger_Meta_Data”]
                                   [UCPTname=“Net/LON/iLON App/Data Logger[0]”]
                </UCPTmetaDataPath>
            </Item>
            <Item xsi:type="UFPTdataLogger_Data" >
                <UCPTname>Net/LON/iLON App/Digital Input 1/nvoClsValue_1</UCPTname>
                <UCPTaliasName>NVL_nvoClsValue_1</UCPTaliasName>
                <UCPTlastUpdate>2008-04-08T11:22:52.952-07:00</UCPTlastUpdate>
                <UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch">0.0 0</UCPTvalue>
                <UCPTvalue LonFormat="UCPTvalueDef">OFF</UCPTvalue>
                <UCPTpointStatus LonFormat="UCPTpointStatus">AL_NO_CONDITION</UCPTpointStatus>
                <UCPTpriority>255</UCPTpriority>
                <UCPTmetaDataPath>//*[@xsi:type=“UFPTdataLogger_Meta_Data”]
                                   [UCPTname=“Net/LON/iLON App/Data Logger[0]”]
                </UCPTmetaDataPath>
            </Item>
      </iLonItem>
    </Read>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

3 SmartServer Applications and the SOAP/XML Interface

This chapter provides an overview of the applications supported by the SmartServer, 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 SmartServer Applications*. This section provides a description of each of the applications that the SmartServer supports.
- *SmartServer XML Configuration Files*. The configuration of each SmartServer application is stored in an XML file. This section lists those XML files, and it indicates where they are stored on the SmartServer.
- *SmartServer Resource Files*. The SmartServer resource files contain information you will need when using the SOAP functions. This section describes how to use the resource files.
- *SOAP Functions*. The SmartServer's SOAP interface includes a generic set of *List*, Get, *Set*, *Read*, *Write*, and *Delete* functions that can be used by the SmartServer applications. Together, these functions make up a symmetric interface. This section provides an overview of how to use these functions.
- *Performance Issues*. This section lists performance issues you should consider when using the SOAP/XML interface.

3.1 Overview of SmartServer Applications

You can use the SOAP/XML interface to configure the following SmartServer applications:

- *Data Server*. The SmartServer's internal Data Server is a software component that *abstracts* any data element of any bus into a *data point*. This enables the SmartServer's built-in applications and your custom SmartServer Web pages to operate on these abstractions without regard of the device driver (e.g., LONWORKS, Modbus, M-Bus, Virtual, and Freely Programmable Module [FPM]).
- *Web Binding*. You can use the Web Binder application to create Web connections that allow direct data exchange over a TCP/IP network between a SmartServer and another host device such as a remote SmartServer, LNS Server, or a WebBinder Target Server (a Web server that can process SOAP requests such as Apache or IIS).
- *Data Logging*. You can configure the SmartServer 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 log entries for each of the updates to the data points it is monitoring. These logs can be downloaded and read with the Internet File Transfer Protocol (FTP), or retrieved with the *Read* function.
- *Alarming*. You can configure the SmartServer to trigger alarms based on the values and statuses of the data points in your control network. The SmartServer 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.
- *Analog Functional Blocks*. You can use the Analog Functional Block application to perform statistical operations on the values of any of the data points in your network.
- *Scheduling*. You can use the SmartServer 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, M-Bus or Modbus device. You can create Schedulers and Calendars to

manage these tasks. You can also create a Real-Time Cock to create events based on sunrise and sundown times.

- *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.

3.2 SmartServer XML Configuration Files

The configuration of each SmartServer application is stored in an XML file. You will use the following XML files to configure the applications of your SmartServer:

```
/root/config/network/<network>/<channel>/i.LON App||<device>/
#8000010128000000[4].UFPTdataLogger.xml

/root/config/network/<network>/<channel>/i.LON App||<device>/
#8000010128000000[4].UFPTalarmNotifier.xml

/root/config/network/<network>/<channel>/i.LON App||<device>/
#8000010128000000[4].UFPTalarmGenerator.xml

/root/config/network/<network>/<channel>/i.LON App||<device>/
#8000010128000000[4].UFPTanalogFunctionBlock.xml

/root/config/network/<network>/<channel>/i.LON App || <device>/
#8000010128000000[4].UFPTscheduler.xml

/root/config/network/<network>/<channel>/i.LON App || <device>/
#8000010128000000[4].UFPTcalendar.xml

/root/config/network/<network>/<channel>/i.LON App || <device>/
#8000010128000000[4].UFPTrealTimeClock.xml

/root/config/network/<network>/<channel>/i.LON App || <device>/
#8000010128000000[4].UFPTtypeTranslator.xml
```

The `/root/config/software` directory includes a sub-directory called TranslatorRules, which contains several XML files you can use when configuring your Type Translators.

Note: The `/root/config/software` directory also contains a file called RNI.xml, which contains configuration data used by the SmartServer remote network interface (RNI), and a file called LSPA.xml, which contains configuration data used when the SmartServer connects to the LonScanner™ Protocol Analyzer. There is no SOAP interface for these applications, and you should not manually edit the RNI.xml or LSPA.xml files. You can configure the RNI application using the SmartServer Web pages. For more information on this, see the *i.LON SmartSever User's Guide*. For more information on the LonScanner Protocol Analyzer, see the *LonScanner Protocol Analyzer User's Guide*.

3.2.1 Modifying the XML Configuration Files

Each application includes a *Set* function. You can use the *Set* function to create and write to the XML file for that application. The SmartServer will modify the XML file, and the operating parameters of the associated application, each time it receives a *Set* message.

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 SmartServer via FTP. Echelon does not recommend this, as you will need to reboot the SmartServer for it to read the downloaded files, and the SmartServer will not perform error-checking on the downloaded XML files.

3.3 SmartServer Resource Files

There are many configuration properties you can configure with 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 SmartServer resource files. In order to successfully send a SOAP message to the SmartServer, all data in the message must be formatted as described in this document and in the resource files.

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

You can use the Node Builder Resource Editor, which is included on the SmartServer software 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\YourCompany\YourResourceFiles.*
```

When adding these files to the SmartServer, it is the best practice to FTP them to a location on the SmartServer flash disk matching the path on your computer:

```
/root/lonworks/types/User/YourCompnay/YourResourceFiles.*
```

You only need to FTP your own custom resource files to the SmartServer. 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 SmartServer you must reboot to be able to use the new type definitions and formats. During boot the SmartServer reads the resource files in this directory and updates its local catalog accordingly.

3.3.1 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 SmartServer, you can find these files in the `/root/lonworks/types` directory. 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 SmartServer, you will assign that data point a format type. If a specific SNVT format is desired for a particular data point, the <UCPTformatDescription> of that data point must be set to the name of that SNVT format.

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

3.3.2 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 SmartServer, these files can be found in the `directory /root/lonworks/types` directory. These files are named STANDARD.ENU, STANDARD.TYP, STANDARD.FMT and STANDARD.FPT.

Many configuration properties that are used by the SmartServer 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 SmartServer.

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

3.3.3 User-Defined 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 SmartServer, you can find all device resource files in the /root/lonworks/types directory.

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:

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

3.3.4 User-Defined 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 SmartServer, these files may be found in the `/root/lonworks/types` directory. The files are named BAS_Controller.ENU, BAS_Controller.TYP, BAS_Controller.FMT and BAS_Controller.FPT.

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

3.3.5 Data Point Templates

There is a set of data point templates in the root/config/template/lonworks/Dp directory that provide the default configurations for new instances of specific data point types. These XML files contain the default presets and the default heartbeat, throttle, and offline configuration properties for a number of common data types, including **SNVT_switch**, **SNVT_occupancy**, **SNVT_temp** (including #SI and #US formats), and **SNVT_temp_f** (including #SI and #US formats). This is very useful for working with the data points of the external devices that you may add to the SmartServer.

3.3.6 Data Formatting

In order to keep the 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 SmartServer. The one exception is the support of measurement system locale, which was introduced in version 1.1 of the SOAP/XML interface. The following list describes the various regional settings used by the SmartServer 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 vertical bar "|", the *i*.LON SmartServer 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 SmartServer 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 a data point *Read* function will be in Fahrenheit.

If that data point is an input to the Alarm Generator, 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 a *Get* function. Furthermore, you would have to use US units when setting the property with a *Set* 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 SmartServers. 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.

3.4 SOAP Functions

The SOAP interface includes a generic set of *List*, Get, *Set*, and *Delete* functions that can be used by the SmartServer applications. Together, these functions make up a symmetric interface. You can use the response from the *List* function as the input to the *Get* function. You can use the response from the *Get* function as the input to the *Set* function.

The SOAP interface also includes *Read* and *Write* methods that can be use to read and write values to the data points on the Data Server, read entries in a data log, alarm log, or scheduler or calendar event log, and update entries in an alarm log. The *Read* and *Write* functions also provide a symmetric interface. You can use the output from the *Read* function as the basis for your input to the *Write* function.

This section provides an overview of the functions in the SOAP interface, and it briefly describes how you can use them.

3.4.1 List Functions

Use the *List* function to retrieve the names of the application instances or items of a specific type on the SmartServer. For example, the *List* function can return a list containing the names of the Alarm Generators, Data Loggers, Schedulers, or other functional blocks representing the SmartServer's embedded applications that have been instantiated. Similarly, you can use the *List* function to return a list of the data points on the Data Server (Dp_Cfg), or retrieve a list of the channels (Channel_Cfg), the LONWORKS application devices (LON_Device_Cfg), the Modbus devices (MOD_Device_Cfg), and other network items on the SmartServer.

3.4.2 Get Functions

You can use the *Get* function to retrieve the configuration of any application instance or item that you have added to the SmartServer. For example, you could use the *Get* function to retrieve the configurations of the Alarm Generators, Data Loggers, Schedulers, and other application instances that have been added to the SmartServer. Similarly, you could use the *Get* function to retrieve the configurations of the data points on the Data Server (Dp_Cfg), or retrieve the LONWORKS application devices (LON_Device_Cfg), the Modbus devices (MOD_Device_Cfg), and other network items on the SmartServer. Note that you must reference the item whose configuration is to be retrieved by its <UCPTname>.

Consider a scenario where you have used a *List* function to retrieve a list containing the <UCPTname> of each Alarm Generator that has been added to the SmartServer. You could use the list as the input for the *Get* function. The *Get* function would return the configuration of all the Alarm Generators included in the list. You can also use the *Get* function to retrieve the configuration of a single Alarm Generator, by supplying the <UCPTname> of the Alarm Generator functional block as the input.

3.4.3 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/network/<network>/<channel>/iLONApp ||<device> directory on the SmartServer, if it has not already been created. All data defined in the input supplied 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 *Set* function to create an Alarm Generator functional block, the #8000010128000000[4].UFPTalarmGenerator.xml will be created in the root/config/network/<network>/<channel>/iLONApp ||<device> directory of the SmartServer (if it does not already exist based on the creation of another alarm generator functional block instance). The file will contain an element for each Alarm Generator defined in the input passed to the function.

After its initial invocation, you can use *Set* function to overwrite the values of the 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 existing Alarm Generators.

When using the *Set* function to create an item such as an Alarm Generator, you should 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 SmartServer applications.

1. Invoke the *List* function to generate a list of Alarm Generators that have been added to the SmartServer. This list includes the <UCPTname> of each Alarm Generator.
2. Invoke the *Get* function, using the list returned by the *List* 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 *Get* 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 response returned by *Get* to match the configuration you want for the new Alarm Generator. This will be more efficient than building the input for the Set function from scratch.

   Note: You must change the <UCPTname> of the Alarm Generator when using this algorithm. Otherwise, the next step of this procedure will overwrite the configuration of the default Alarm Generator you have chosen.
4. Invoke the *Set* function, using the modified response from Step 3 as input. The new item is successfully created, without recreating an input that defines an entire Alarm Generator configuration from scratch, and with minimal risk of format errors. Chapters 4-12 will clarify the benefits of this algorithm.

3.4.4 Read Functions

You can use the *Read* function to read the value, status, or priority of a data point on the Data Server (Dp_Cfg), read the entries in an alarm log or data log, and read the events scheduled in a Scheduler or Calendar. To use the *Read* function, you need to provide the <UCPTname> of the item to be read. This could be a data point, or an Alarm Notifier, Data Logger, Scheduler, or Calendar functional block. The *Read* function returns a list of <Item> elements of a Dp_Data type for each data point referenced by the input you supplied to the function.

3.4.5 Write Functions

You can use the *Write* function to update the value, priority, or status of a data point on the Data Server or to update an entry in an Alarm Log. To use the *Write* function, you need to provide an <Item> element of a Dp_Data type that includes the <UCPTname> of the item to be updated, which could be a data point or an Alarm Notifier functional block. When using the *Write* function, you can use the output supplied by the corresponding *Read* function as the basis for your input.

3.4.6 Delete Functions

Use the Delete functions to delete items from an application. For example, you can use the *Delete* function to delete an Alarm Generator functional block or to delete a data point on Data Logger. To delete an item, you must provide an <Item> element of a specific xsi:type (corresponding to the item's driver) that includes the <UCPTname> of the item to be deleted.

3.5 Performance Issues

The SmartServer contains 64 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 SmartServer, especially using the SOAP interface, to eventually exhaust its memory. This could result in delays in network access of the SmartServer, performance problems for the SmartServer applications, or in the worst case even a reboot of the SmartServer.

If your SmartServer 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 SmartServer's SOAP interface. While they are not absolute limits or guarantees of performance, they may be helpful to follow when attempting to manage the SmartServer'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.
- 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 Chapter 7.
- 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 Chapter 5.
- If the combined XML file sizes for a given application exceed 100 KB, do not try to read all the configuration data for that application in a single *Get* message. This could potentially happen with the Scheduler application if all of its functional blocks were used, or possibly with the Alarm Notifier application.
- Do not send a request message larger than 100 KB. Some possible examples of this might be using a single Set message to define more than 100 data points or write to 40 Alarm Notifiers.
- Limit the number of simultaneous SOAP clients to no more than the number of Web tasks specified in the **WebParams.dat** file on the SmartServer, which is eight. Also note that you should not open more than two or three Web browsers for a given SmartServer.

4 Using the SmartServer Data Server

The SmartServer's internal Data Server is a software component that *abstracts* any data element of any bus into a *data point*. This enables the SmartServer's built-in applications and your custom SmartServer Web pages to operate on these abstractions without regard of the device driver (e.g., LONWORKS, Modbus, M-Bus, Virtual, and Freely Programmable Module [FPM]).

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

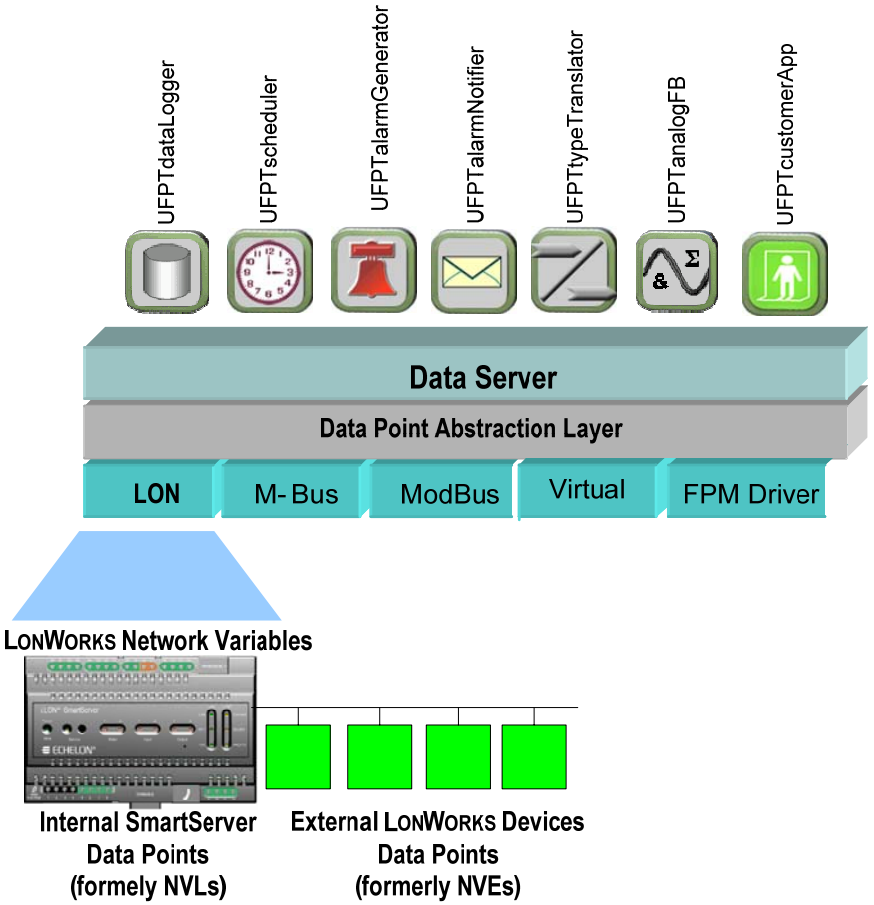
At the Data Server layer, all data points have the same set of properties, regardless of the network or device to which each data point is local. This is made possible by the drivers that exist for each data point type, which handle communication between the Data Server and the physical network to which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

You can use a standard network management tool for the particular data point type to configure each driver on the SmartServer. For example, you could use an LNS-based network management tool to configure the data points on the SmartServer's internal automated systems device (i.LON App). This layer of abstraction between the drivers and the Data Server provides a mechanism for all SmartServer 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 SmartServer remain synchronized with each other, and within the device to which the data point is local. The tools you can use to configure the SmartServer include custom SOAP applications, the LonMaker tool, built-in SmartServer Web pages, and custom SmartServer Web pages.

The data elements that that can be abstracted by the Data Server include the following:

- The network variables and configuration properties on the SmartServer's LON driver, which include the following:
  - The network variables and configuration properties on the SmartServer's internal automated systems device (Net/LON/i.LON App). The i.LON App device contains functional blocks representing the SmartServer's built-in applications. These data points were referred to as NVLs in Version 3.0.
  - The network variables and configuration properties of the LONWORKS devices connected to the SmartServer. These data points were referred to as NVEs in Version 3.0.
  - The system network variables on the SmartServer that maintain constant values for the SmartServer's built-in applications (e.g., Net/LON/iLON App/Alarm Generator[0]/CompareDP). These data points were referred to as NVCs in Version 3.0.
- The registers of M-Bus and Modbus devices on the SmartServer's M-Bus and Modbus drivers.
- The network variables on the SmartServer's Virtual driver. This includes the network variables of the SmartServer's internal systems device (Net/VirtCh/i.LON System), which contain connection manager and LonTalk statistics, and the network variables on Interoperable Self-Installation (ISI) devices connected to the SmartServer. These data points were referred to as NVVs in Version 3.0.
- The network variables and configuration properties on custom FPM drivers created with the *i*.LON SmartServer Programming Tools. You can use FPM drivers as gateways to legacy systems or nonnative networks such as BACnet and CAN (requires an external interface, sold separately). For more information on creating FPM drivers, see the *i.LON SmartServer Programming Tools User's Guide*.

The following figure shows the relationship between the buses supported by the SmartServer, the Data Server, and the SmartServer's built-in applications.



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 SmartServer 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 Data Server. Once you have created the data points for your SmartServer and added them to the Data Server, you can reference these data points when using SmartServer applications such as the Analog Function Block, Scheduler, 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. Echelon recommends that you restrict all networks to a maximum of 1,000 data points.

4.1 Creating and Modifying the Data Point XML Files

The SmartServer stores the data points on the Data Server in Dp.XML files under the parent network/channel/device directory of the data points. Consider the following examples:

- The data points on the SmartServer's internal automated systems device (i.LON App), which contains the functional blocks representing the SmartServer's built-in applications, are stored in a Dp.xml file in the root/config/network/Net/LON/iLONApp directory on the SmartServer flash disk by default.
- The data points for an external LONWORKS digital input/output device connected to the SmartServer might be stored in a Dp.xml file in the root/config/network/Net/LON/DIO-2 directory on the SmartServer flash disk.
- The data points on the SmartServer's internal systems device (i.LON System), which contains Interoperable Self-Installation (ISI) data points and data points containing connection manager and LonTalk statistics, are stored in a Dp.xml file in the root/config/network/Net/VirtCh/iLONSystem directory on the SmartServer flash disk.
- **Note:** The directories containing the Data Server's Dp.xml files also contain separate <driver>_Dp.xml files in which the data points' driver properties are stored. The properties in these files relate to a specific bus—LONWORKS, Modbus, M-Bus, Virtual, or FPM—and not the Data Server. These driver-specific properties and how to use the *List*, *Get*, *Set*, and *Delete* functions on them are described in Chapters 14–17.

You can manage the `Dp.xml` files manually using an XML text editor, and download them to `the root/config/network/<network>/<channel>/<device>` directory of the SmartServer via FTP. Echelon does not recommend this, as the SmartServer will require a reboot to read the configuration of the downloaded XML files. Additionally, the SmartServer 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.

4.2 Overview of the Data Point XML File

The Dp.xml files on the SmartServer store the configurations of the data points that you have added to the SmartServer under their respective parent network/channel/device directories. Each data point is signified by an <Item> element of a Dp_Cfg type in the XML file. The configuration properties contained in each <Item> element define the configuration of a data point, and are described later in this chapter.

You can create new data points using the *Set* function, or by manually editing the Dp.xml file. You can read the current value and status of a data point using the *Read* function, and you can write updated values to data point using the *Write* function.

The following code represents a snippet of a sample Dp.xml file with one data point of an external LONWORKS digital input/output device connected to the SmartServer:

```
<iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="Dp_Cfg">
        <UCPTname>Net/LON/DIO-3/Digital Output[0]/DO_Digital_1</UCPTname>
        <UCPTannotation>Dp_In;xsi:type="LON_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-17T16:00:02.202-07:00</UCPTlastUpdate>
        <UCPTdescription>Digital value to output</UCPTdescription>
        <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
        <UCPTlength>2</UCPTlength>
        <UCPTdirection LonFormat="UCPTdirection">DIR_IN</UCPTdirection>
        <UCPTunit field="value">% of full level</UCPTunit>
        <UCPTunit field="state">state code</UCPTunit>
        <UCPTbaseType LonFormat="UCPTbaseType">BT_STRUCT</UCPTbaseType>
        <UCPTmaxFields>2</UCPTmaxFields>
        <SCPTmaxSendTime>0.0</SCPTmaxSendTime>
        <SCPTmaxRcvTime>0.0</SCPTmaxRcvTime>
```

```
        <ValueDef>
            <UCPTindex>0</UCPTindex>
            <UCPTname>OFF</UCPTname>
            <UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch">0.0 0</UCPTvalue>
        </ValueDef>
        <ValueDef>
            <UCPTindex>1</UCPTindex>
            <UCPTname>ON</UCPTname>
            <UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch">100.0 1</UCPTvalue>
        </ValueDef>
    </Item>
</iLonItem>
```

4.3 Data Server SOAP Interface

You can use the SOAP interface to perform the following functions on a data point on the Data Server.

Function	Description
List	List the name of each data point that you have added to the Data Server.
Get	Retrieve the configuration of a data point.
Set	Create a data point and add it to the Data Server, or modify the configuration of an existing data point.
Read	Read the current values, statuses, and priorities of one or more data points.
Write	Write to the current values, statuses, and priorities of one or more data points.
Invoke	Used with *Dp_ResetPrio_Invoke* xsi type to reset the priority level assigned to a data point.
Delete	Remove a data point from the Data Server.

Note: Section 21.1.1, *Reading and Writing Data Point Values in Visual C# .NET*, includes a C#.NET programming example demonstrating how to use the Data Server SOAP interface to read and write data point values. Section 21.2.1, *Reading and Writing Data Point Values in Visual Basic.NET*, includes a Visual Basic .NET example demonstrating how to do this.

Section 22.3.1, *Reading and Writing Data Point Values in Java*, includes a Java example demonstrating how to read and write data point values.

4.3.1 Using the List Function on the Data Server

You can use the *List* function to retrieve a list of data points that have been added to the Data Server. The *List* function takes an <iLonItem> element that has an xSelect statement querying items of a Dp_Cfg type as its input, as shown in the example below. The *List* function returns an <Item> element for each data point that you have added to Data Server.

Request (all data points on the Data Server)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="Dp_Cfg"]</xSelect>
  </iLonItem>
</List>
```

You can use additional filters in the xSelect statement to return a specific set of data points on the Data Server. These filters include the data point's <UCPTname> and <UCPTlastUpdate> properties, and its position on the data server.

Request (return all data points on the internal SmartServer automated systems device [i.LON App])

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="Dp_Cfg"]
                  [starts-with(UCPTname,"Net/LON/iLON App")]
    </xSelect>
  </iLonItem>
</List>
```

Request (return the first 5 data points on the Net/LON/DIO-1/Digital Encoder functional block that have been updated since 2008-03-17T00:00:00)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="Dp_Cfg"][position()&lt;5]
                  [starts-with(UCPTname,"Net/LON/DIO-1/Digital Encoder")]
                  [UCPTlastUpdate&gt;"2008-03-17T00:00:00"]
    </xSelect>
  </iLonItem>
</List>
```

Response

```
<ListResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
    <Item>
        <UCPTname>Net/LON/DIO-3/Digital Encoder[0]/DE_D1_1</UCPTname>
        <UCPTannotation>Dp_In;xsi:type="LON_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
    <Item>
        <UCPTname>Net/LON/DIO-3/Digital Encoder[1]/DE_D1_2</UCPTname>
        <UCPTannotation>Dp_In;xsi:type="LON_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
    <Item>
        <UCPTname>Net/LON/DIO-3/Digital Encoder[0]/DE_D2_1</UCPTname>
        <UCPTannotation>Dp_In;xsi:type="LON_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
    <Item>
        <UCPTname>Net/LON/DIO-3/Digital Encoder[1]/DE_D2_2</UCPTname>
        <UCPTannotation>Dp_In;xsi:type="LON_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
    <Item>
        <UCPTname>Net/LON/DIO-3/Digital Encoder[0]/DE_D3_1</UCPTname>
        <UCPTannotation>Dp_In;xsi:type="LON_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
  </iLonItem>
</ListResponse>
```

4.3.2 Using the Get Function on the Data Server

You can use the *Get* function to retrieve the configuration of any data point that you have added to the SmartServer's Data Server. The input parameters you supply to the function will include one or more <Item> elements. Each <Item> element includes a <UCPTname> property that you can use to specify the data point to be returned, as well as any number of child elements you can use to identify the data points whose configurations are to be returned.

Alternatively, you can specify one or more data point properties such as <UCPTformatDescription>in an xSelect statement to filter the items returned by the *Get* function.

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

Request (return all SNVT_switch data points on the internal SmartServer automated systems device [i.LON App])

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="Dp_Cfg"]
                  [starts-with(UCPTname,"Net/LON/iLON App")]
                  [UCPTformatDescription="#0000000000000000[0].SNVT_switch"]
    </xSelect>
  </iLonItem>
</List>
```

Request (return a specific data point)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>Net/LON/DIO-3/Digital Encoder[0]/DE_D1_1</UCPTname>
    </Item>
  </iLonItem>
</Get>
```

Response

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="Dp_Cfg">
        <UCPTname>Net/LON/DIO-3/Digital Encoder[0]/DE_D1_1</UCPTname>
        <UCPTannotation>Dp_In;xsi:type="LON_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-17T13:36:48.460-07:00</UCPTlastUpdate>
        <UCPTdescription>Digital encoder input 1.  This is the least-significant
bit</UCPTdescription>
        <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
        <UCPTlength>2</UCPTlength>
        <UCPTdirection LonFormat="UCPTdirection">DIR_IN</UCPTdirection>
        <UCPTunit field="value">% of full level</UCPTunit>
        <UCPTunit field="state">state code</UCPTunit>
        <UCPTbaseType LonFormat="UCPTbaseType">BT_STRUCT</UCPTbaseType>
        <UCPTmaxFields>2</UCPTmaxFields>
        <SCPTmaxSendTime>0.0</SCPTmaxSendTime>
        <SCPTmaxRcvTime>0.0</SCPTmaxRcvTime>
        <ValueDef>
            <UCPTindex>0</UCPTindex>
            <UCPTname>OFF</UCPTname>
            <UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch">0.0 0</UCPTvalue>
        </ValueDef>
        <ValueDef>
            <UCPTindex>1</UCPTindex>
            <UCPTname>ON</UCPTname>
            <UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch">100.0 1</UCPTvalue>
        </ValueDef>
    </Item>
<iLonItem>
```

The *Get* function returns an <Item> element for each data point referenced in the input parameters you supplied to the function. The properties included within each <Item> element are initially defined when the data point is added to the DataServer. You can write to these data point properties with the *Set* function. The following table describes these properties.

Property	Description
<UCPTname>	The name of the data point in the following format: <*network*/*channel*/*device*/*functional block/data point*>.
<UCPTannotation>	The direction of the data point (Dp_In, Dp_Out, or Dp_In_Out [unspecified]), and its xsi type, which corresponds to the bus on which the data point resides (e.g., LON_Dp_Cfg for a LONWORKS data point) . This determines the icon used to represent the data point in the navigation pane on the left side of the SmartServer Web interface.
<UCPThidden>	A flag indicating whether the data point is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **0** – shown **1** – hidden
<UCPTlastUpdate>	A timestamp indicating the last time the configuration of the data point was updated. This timestamp uses the ISO 8601 format, which is as follows: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Data Point was last updated. The second segment (after the **T):** HH:MM:SS.sss represents the time of day the configuration of the Data Point 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 be expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. It can be left out. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002. If it is not UTC, time shift has to be defined: 2008-02-28T09:59:53.660+01:00
<UCPTdescription>	A user-defined description of the data point. This can be a maximum of 201 characters long.
<UCPTformatDescription>	The data point's program ID; data type (SNVT, SCPT, UNVT, UCPT, or built-in data type); and format (e.g., SI metric or US customary if the type has multiple formats such as **SNVT_temp_p**). The format description is displayed in the following format: #<*manufacturer ID*>**[***scope selector***].**<*type name*>[#format] . This determines many factors about the data point, including the type of values it takes and its base type. This could be any

Property	Description
	standard (SNVT) format type included in the resource files on the SmartServer, or any user-defined (UNVT) format type included in resource files uploaded to the SmartServer. For more information on the resource files, see *SmartServer Resource Files*. If you do not set this property, it is set to RAW_HEX and the data point uses raw hex values. The SNVT format types included in the SmartServer resource files are also listed and described in the SNVT Master List, which can be downloaded from Echelon's Support Web site at: *www.echelon.com*
<UCPTlength>	Specifies the size (in bytes) of the data point.
<UCPTdirection>	Specifies whether the data point is an input data point (DIR_IN), output data point (DIR_OUT), or has an unspecified direction (DIR_NUL).
<UCPTpersist>	A flag indicating that the value stored in the data point persists through SmartServer reboots. If this property is set to 1, the last data point value is stored in the <UCPTdefOutput> property. Configuration properties are marked as persistent by default.
<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 slave devices, 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 the *Set* function to change this value in the Data Server. However, you must program your application to enforce the new value, as the SmartServer will continue to enforce the default value taken from the resource files.
<UCPTunit>	This property is a string up to 227 characters long that describes the units the value in which a data point is measured. It is 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 resource files on the SmartServer. For scalar and enumerated data points, this property specifies the units of measures used by the data point. For example, the unit string of a **SNVT_temp_f** data point is **°F**. The unit string is defined by resource files. For structured data points, the fields within the data point are specified in a series of <UCPTunit> properties if the unit strings can be edited. Using a **SNVT_switch** data point for example, **value** and **state** fields will be specified in a series of <UCPTunit> properties with their respective units of measure ("% of full level" and "state code"). You can use the *Set*

Property	Description
	function to edit the unit strings of data point fields.
<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 SmartServer.
<UCPTmaxFields>	This property specifies the maximum number of fields that the data point may have. For scalar and enumerated data point this property is 0 if the unit string cannot be edited, or it is 1 if the unit string is editable. For structured data points, this property is 2 or more based on the SNVT or UNVT used by the data point. For example, the default value in <UCPTmaxFields> for a **SNVT_switch** data point will be 2 (value and state), and it will be 3 for a **SNVT_setting** data point (function, setting, and rotation).
<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 SmartServer 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.
<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.
<SCPTmaxRcvTime>	This property is used to control the maximum time that can elapse after an update to a bound network input before another update can occur. 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 7, *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.
<UCPTminValue>	Optional. This value is initially taken from the resource files, if

Property	Description
	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 the *Set* function to change this limit in the Data Server. However, you must program your application to enforce the new limit, as the SmartServer will continue to enforce the limit taken from the resource files.
<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 the *Set* function to change this limit in the Data Server. However, you must program your application to enforce the new limit, as the SmartServer 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 to this value, the <UCPTpointStatus> of the data point will be set to AL_VALUE_INVALID. The status will be returned to a normal condition (AL_NO_CONDITION) as soon as the value is set to a valid value again. A default value will be assigned to the <UCPTinvalidValue> property based on the <UCPTformatDescription> selected, if an invalid value is defined for the selected format in the resource files. You could create an Alarm Notifier to trigger an alarm notification when an invalid value is written to a data point and the data point's status is updated to AL_VALUE_INVALID. For more information on Alarm Notifiers, see Chapter 7, *Alarm Notifier*.
<ValueDef>	The <ValueDef> elements specify preset value definitions that can be assigned to the data point. You can use these preset values to update the value of the data point when other SmartServer applications such as the Event Scheduler or the Alarm Notifier reference them. Each <ValueDef> element includes three properties: • <UCPTindex>. The index value assigned to the preset. • <UCPTname>. The name of the preset. You will use this name when referencing the preset value with other applications. • <UCPTvalue>. The value the data point should be assigned to when this preset is used. The values entered here must be in valid format, as defined by the network variable type assigned to the data point.

4.3.3 Using the Set Function on the Data Server

You can use the *Set* function to overwrite the configuration of a data point, or to create a new data point and add it to the Data Server. The input parameters you supply to the function will include one

or more <Item> elements. Each <Item> element includes a <UCPTname> property that specifies a unique data point to be created or modified.

Each <Item> element may also include a series of properties that define the configuration of the new (or modified) 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, *Using the Get Function on the Data Server*, details the properties you can include in the *Set* function.

You can set multiple data points with a single *Set* message. However, you should not attempt to create or write to more than 100 data points with a single call to the *Set* function. Additionally, to optimize the memory available to the SmartServer, you should not have more than 1,000 data points in your network at any time.

The following example demonstrates how to add a new **SNVT_temp f [1]** input data point to the data server on the Net/LON/iLON App/VirtFB functional block.

Request

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="Dp_Cfg">
        <UCPTname>Net/LON/iLON App/VirtFb/temp_f[1]</UCPTname>
        <UCPTannotation>Dp_In</UCPTannotation>
        <UCPTformatDescription>#0000000000000000[0].SNVT_temp_f#US</UCPTformatDescription>
        <UCPTdirection LonFormat="UCPTdirection">DIR_IN</UCPTdirection>
        <UCPTpersist>1</UCPTpersist>
        <UCPTdefOutput LonFormat="#0000000000000000[0].SNVT_temp_f#US">72.5</UCPTdefOutput>
        <UCPTunit field="">°F</UCPTunit>
        <ValueDef>
            <UCPTindex>0</UCPTindex>
            <UCPTname>OCCUPIED</UCPTname>
            <UCPTvalue LonFormat="#0000000000000000[0].SNVT_temp_f#US">69.8</UCPTvalue>
        </ValueDef>
        <ValueDef>
            <UCPTindex>1</UCPTindex>
            <UCPTname>UNOCCUPIED</UCPTname>
            <UCPTvalue LonFormat="#0000000000000000[0].SNVT_temp_f#US">60.8</UCPTvalue>
        </ValueDef>
        <ValueDef>
            <UCPTindex>2</UCPTindex>
            <UCPTname>STANDBY</UCPTname>
            <UCPTvalue LonFormat="#0000000000000000[0].SNVT_temp_f#US">66.2</UCPTvalue>
        </ValueDef>
    </Item>
  </iLonItem>
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/LON/iLON App/VirtFb/temp_f[1]</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

4.3.4 Using the Read Function on the Data Server

You can use the *Read* function to read the value and status of any data point that you have added to the Data Server. The input parameters you supply to the function will include one or more <Item> elements. Each <Item> element includes a <UCPTname> property that specifies the data points whose values and statuses are to be returned.

Alternatively, you can use an xSelect statement to return the values and statuses of a specific set of data points on the Data Server. The filters you can use in an xSelect statement in the *Read* function

include the data point's <UCPTname> and <UCPTlastUpdate> properties and the position on the SmartServer. You can also use the <UCPTvalueFormat> in an xSelect statement to specify that the *Read* function return the data point values in raw hex or return the values of the fields within structured data points individually.

Note: You cannot use the position () filter with in an xSelect statement that includes the <UCPTlastUpdate> filter. If you write an xSelect statement with both of these filters, the position () filter is ignored.

The *Read* function will return a list of <Item> elements of a Dp_Data type for each data point referenced by the input you supplied to the function. Each of these <Item> elements contains the current values of a group of properties and attributes associated with the referenced data point. This includes the value, status, and the current priority level of the data point.

Request (a specific data point on the Net/LON/DIO-1/Digital Encoder functional block)

```
<Read xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>Net/LON/DIO-3/Digital Encoder[0]/DE_D1_1</UCPTname>
    </Item>
  </iLonItem>
</Read>
```

Request (use an xSelect statement to return all data points on the Data Server)

```
<Read xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="Dp_Cfg"]</xSelect>
  </iLonItem>
</Read>
```

Request (use an xSelect statement to return the first 50 data points on the internal SmartServer automated systems device [i.LON App])

```
<Read xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="Dp_Cfg"]
                  [starts-with(UCPTname,"Net/LON/iLON App")]
                  [position()&lt;50]
    </xSelect>
  </iLonItem>
</Read>
```

Request (use an xSelect statement to return the data points on the Data Server that have been updated since 2008-03-17T00:00:00 with raw hex values)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="Dp_Cfg"]
                  [UCPTlastUpdate&gt;"2008-03-17T00:00:00"]
                  [UCPTvalueFormat="VF_RAW_HEX"]
    </xSelect>
  </iLonItem>
</List>
```

Request (use an xSelect statement to return the data points on the Net/LON/i.LON App/VirtFb functional; individually list values of fields within structured data points)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="Dp_Cfg"]
                  [starts-with(UCPTname,"Net/LON/iLON App/VirtFb")]
                  [UCPTvalueFormat="VF_FIELDS"]
    </xSelect>
  </iLonItem>
</List>
```

Response

```
<ReadResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="Dp_Data" >
        <UCPTname>Net/LON/iLON App/VirtFb/temp_thermostat</UCPTname>
        <UCPTannotation>Dp_In;xsi:type="LON_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-14T16:34:11.310-07:00</UCPTlastUpdate>
        <UCPTvalue LonFormat="#0000000000000000[0].SNVT_temp" Unit="°C">21.0</UCPTvalue>
        <UCPTvalue LonFormat="UCPTvalueDef">OCCUPIED</UCPTvalue>
        <UCPTpointStatus LonFormat="UCPTpointStatus">AL_NUL</UCPTpointStatus>
        <UCPTpriority>255</UCPTpriority>
    </Item>
    <Item xsi:type="Dp_Data" >
        <UCPTname>Net/LON/iLON App/VirtFb/nviSwitch</UCPTname>
        <UCPTannotation>Dp_In;xsi:type="LON_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-14T16:34:11.340-07:00</UCPTlastUpdate>
        <UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch" Unit="value, state">0.0 0</UCPTvalue>
        <UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch.value" Unit="% of full level" >0.0</UCPTvalue>
        <UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch.state" Unit="state code" >0</UCPTvalue>
        <UCPTvalue LonFormat="UCPTvalueDef">OFF</UCPTvalue>
        <UCPTpointStatus LonFormat="UCPTpointStatus">AL_NUL</UCPTpointStatus>
        <UCPTpriority>255</UCPTpriority>
    </Item>
    <Item xsi:type="Dp_Data" >
        <UCPTname>Net/LON/iLON App/VirtFb/nvoSetting</UCPTname>
        <UCPTannotation>Dp_Out;xsi:type="LON_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-14T16:34:11.380-07:00</UCPTlastUpdate>
        <UCPTvalue LonFormat="#0000000000000000[0].SNVT_setting" Unit="function, setting, rotation"
        >SET_OFF 0.0 0.00</UCPTvalue>
        <UCPTvalue LonFormat="#0000000000000000[0].SNVT_setting.function" Unit="setting control
        function names">SET_OFF</UCPTvalue>
        <UCPTvalue LonFormat="#0000000000000000[0].SNVT_setting.setting" Unit="% of full level"
        >0.0</UCPTvalue>
        <UCPTvalue LonFormat="#0000000000000000[0].SNVT_setting.rotation" Unit="degrees"
        >0.00</UCPTvalue>
        <UCPTpointStatus LonFormat="UCPTpointStatus" >AL_NUL</UCPTpointStatus>
        <UCPTpriority>255</UCPTpriority>
    </Item>
  </iLonItem>
</ReadResponse>
```

The following table describes the properties of each <Item> element returned by the *Read* function.

Property	Description
<UCPTname>	The name of the data point in the following format: *<network/channel/device/functional block/data point>*.
<UCPTannotation>	The direction of the data point (Dp_In, Dp_Out, or Dp_In_Out [unspecified]), and its xsi type, which corresponds to the bus on which the data point resides (e.g., LON_Dp_Cfg for a LONWORKS data point) . This determines the icon used to represent the data point in the navigation pane on the left side of the SmartServer Web interface.

Property	Description
<UCPThidden>	A flag indicating whether the data point is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **0** – shown **1** – hidden
<UCPTlastUpdate>	A timestamp indicating the last time the configuration of the data point was updated. This timestamp uses the ISO 8601 format, which is as follows: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm
<UCPTvalue>	The current value in the data point. This property may also include the following attributes: • LonFormat (format description). The format defined by the data type used by the data point. This attribute is expressed in the following format: "#<*programID*>[*scope*].<*data type*>". • LonFormat="UCPTvalueDef". The value definition currently being used by the data point. Value definitions represent preset values. They can be created with the **Configure – Data Points** Web page in the SmartServer Web interface, or with the *Set* function. You can use these value definitions to update the value of the data point other SmartServer applications such as the Event Scheduler or the Alarm Notifier. • Unit. The units of measure used by 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 SmartServer. For more information on these applications, see Chapter 6, *Alarm Generator*, and Chapter 7, *Alarm Notifier*.
<UCPTpriority>	The priority level currently assigned to the data point (0-255). The priority level of a data point determines which applications can write to its value. You can modify the value of this property with the *Write* or *Invoke* functions.

4.3.4.1 Setting the Maximum Age of Data Point Values

When you use the *Read* function to read the value of a data point on the Data Server, you can specify a <UCPTmaxAge> property to set the maximum period of time (in seconds) that data point value is cached in the Data Server before it polls the data point and returns an updated value. This enables you to control the amount of traffic that is generated on a specific channel by your SOAP application.

The value to which you set <UCPTmaxAge> is compared to the amount of time a data point value has been cached in the Data Server, which then does the following:

- If <UCPTmaxAge> is less than the period of time the data point value has been cached, the Data Server polls the data point and returns the updated value.
- If <UCPTmaxAge> is greater than the period of time the data point value has been cached, the Data Server returns the cached value.

- If <UCPTmaxAge> is set to **0**, the Data Server returns polls the data point and returns the updated value regardless how current the data point is.
- If <UCPTmaxAge> is disabled, the Data Server returns cached values regardless how old the data point values are. This is the default.

4.3.5 Using the Write Function on the Data Server

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 write to a data point's current value and priority level with the *Write* function. The input parameters you supply to this function will include one or more <Item> elements that have a <UCPTname> property, specifying the unique name of the data point to be written to with the *Write* function. You should not attempt to write to more than 100 data points with a single call to the *Write* function.

You can specify the value to be written to the data point with the <UCPTvalue> property. You can also write an updated value to the data point using a formatted value, a preset, or raw hex. You can also write values to the individual fields of structured data points.

4.3.5.1 Writing Formatted Values to a Data Point

You can use a formatted value to write to a data point. To do this, you must include a LonFormat attribute in the <UCPTvalue> property and set it to the format description of the data point. This attribute indicates how the <UCPTvalue> property should be unformatted by the SmartServer. If the UCPTformatDescription of the data point being written to is SNVT_temp_f#SI, and the *Write* function includes a LonFormat attribute with the value SNVT_temp_f#US, the specified value will be first unformatted using Fahrenheit, before being written to the Data Point, even though the format of the Data Point is normally in Celsius. For example, you could write a value of 32 to a **SNVT_temp** data point in Fahrenheit by specifying the data type's customary units (#US) format, and the value will automatically be converted to 0.0° Celsius.

- <UCPTvalue LonFormat="#0000000000000000[0].SNVT_temp#US">32.0</UCPTvalue>

4.3.5.2 Writing Presets to a Data Point

You can use a preset to write to a data point. To do this, you must include a LonFormat attribute and set it to the <UCPTvalueDef> property. For example, you could write a value of 100.0 1 to a **SNVT_switch** data point by writing the ON preset to the data point.

- <UCPTvalue LonFormat="UCPTvalueDef">ON</UCPTvalue>

Note: If you pass in both the <UCPTvalue> property and a <UCPTvalue> property with a <UCPTvalueDef> LonFormat attribute in a single *Write* function, the <UCPTvalueDef> property will be used to determine the value to assign to the data point, unless it references an invalid value, in which case the <UCPTvalue> property will be used to determine the value to assign to the data point.

4.3.5.3 Writing Raw Values to a Data Point

You can write a raw value to a data point. To do this, you must include a LonFormat attribute and set it to "RAW_HEX".

```
<UCPTvalue LonFormat="RAW_HEX">10</UCPTvalue>
```

4.3.5.4 Writing Values to Structured Data Points

You can write to the individual fields of structured data points. To do this, you must include a LonFormat attribute and set it to the field of the specific data type using the following format: *datatype.field*. For example, to write to the **setting** field of a **SNVT_scene** data point, you would set the LonFormat attribute to "SNVT_scene.setting" and then specify the value to be written to the field.

```
<UCPTvalue LonFormat= "SNVT_scene.function">SC_RECALL</UCPTvalue>
<UCPTvalue LonFormat= "SNVT_scene.sceneNumber">2</UCPTvalue>
```

When you are writing to the fields of structured data points, 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 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 the *Write* function, and do not want to update the structure over the network until all fields have been written by the function.

4.3.5.5 Writing Priority Levels

The priority level specified for each data point is set by the <UCPTpriority> property. You can enter a value between 0-255 as the priority, where 0 represents the highest priority level and 255 represents the lowest priority level. The priority level you specify must be higher than (or equal to) the priority level 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 *Using the Invoke Function to Reset Data Point Priorities*.

4.3.5.6 Data Server Write Function Examples

The following code samples demonstrate how to use the *Write* function on the Data Server.

Request (write a value to a scalar data point normally measured in Celsius using Fahrenheit)

```
<Write xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>Net/LON/iLON App/VirtFb/temp_f</UCPTname>
        <UCPTvalue LonFormat="#0000000000000000[0].SNVT_temp_f#US">68.0</UCPTvalue>
    </Item>
  </iLonItem
</Write>
```

Request (write a value to a structured data point)

```
<Write xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>Net/LON/iLON App/VirtFb/nvoSwitch</UCPTname>
        <UCPTvalue>100.0 1</UCPTvalue>
    </Item>
  </iLonItem>
</Write>
```

Request (write a value to a structured data point with a preset)

```
<Write xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>Net/LON/iLON App/VirtFb/nvoSwitch</UCPTname>
        <UCPTvalue LonFormat="UCPTvalueDef">ON</UCPTvalue>
    </Item>
  </iLonItem>
</Write>
```

Request (write to the field of a structured data point, but don't propagate the value over the LONWORKS channel)

```
<Write xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>Net/LON/iLON App/VirtFb/nvoSwitch</UCPTname>
        <UCPTvalue LonFormat="SNVT_switch.value">85</UCPTvalue>
        <UCPTpropagate>0</UCPTpropagate>
    </Item>
  </iLonItem>
</Write>
```

Response

```
<WriteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="Dp_Data" >
        <UCPTname>Net/LON/iLON App/VirtFb/nvoSwitch</UCPTname>
        <UCPTannotation>Dp_Out;xsi:type="LON_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTvalue LonFormat="SNVT_switch.value">85</UCPTvalue>
    </Item>
  </iLonItem>
</WriteResponse>
```

4.3.6 Using the Invoke Function to Reset Data Point Priorities

You can use the *Invoke* function to reset the priority of a data point to 255—the lowest possible priority level. The input parameters you supply to this function will include one or more <Item> elements of a Dp_ResetPrio_Invoke type. Each <Item> element must include a <UCPTname> property, specifying the unique name of the data point to whose priority is to be reset to 255, and a <UCPTpriority> property, specifying a priority level that is equal to or greater than the current priority assigned to the data point. You should not attempt to write to more than 100 data points with a single call to the *Invoke* function.

The *Invoke* function resets the priority level assigned to each data point referenced in the input parameters to 255. Once the priority level assigned to a data point has been reset to 255, all applications in which the data point is registered are notified, and the next highest-priority application can write values to that data point.

The priority level specified in the input must be equal or higher priority than the current priority assigned to the data point for it to be reset. For more information on priority levels, see *Writing Priority Levels.*

Request (reset the priority of a data point that currently has a priority level of 240)

```
<InvokeCmd xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="Dp_ResetPrio_Invoke">
        <UCPTname>Net/LON/iLON App/VirtFb/nvoSwitch</UCPTname>
        <UCPTpriority>235</UCPTpriority>
    </Item>
  </iLonItem>
</InvokeCmd>
```

Response

```
<InvokeCmdResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/LON/iLON App/VirtFb/nvoSwitch</UCPTname>
    </Item>
  </iLonItem>
</InvokeCmdResponse>
```

If the priority level you specify in the <UCPTpriority> property is lower than the priority currently assigned to the data point, the Response message will include the following fault code:

```
<fault>
    <faultcode faultType="_error">12</faultcode>
    <faultstring xml:lang="en-US">Priority too low to reset priority</faultstring>
</fault>
```

4.3.7 Data Point Values and Priority Levels

As described in the *Using the Write Function on the Data Server* section, you must specify a priority level in the <UCPTpriority> property that is greater than or equal to the priority level used by the last application in order to write an updated value to a data point.

For example, consider a scenario where a SOAP application uses the *Write* function to write to the value of a data point called "nvoValue". Assume that the last application to write to the value of 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 (inclusive) 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. The following describes 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.

Event	Priority Level Assigned	Result of Operation
Power-Up	255	The value of the data point is updated successfully.
Scheduler Updates Data Point	240	The value of the data point is updated successfully, as the priority used by the Scheduler is greater than that assigned to the data point during power-up.
Custom Application Invokes *Write* Function on Data Server	75	The value of the data point is updated successfully, as the priority used in the call to *Write* function is greater than that assigned to the data point by the Scheduler.
Scheduler Updates Data Point	240	The value of the data point is not updated successfully, as the priority used by the Scheduler is less than that used by the last application to update the data point.
Custom Application Calls *Invoke* function to Reset the Data Point Priority on Data Server	245	The priority of the data point is not reset to 255, as the priority level set in the *Invoke* function (245) is less than that used by the last application to update the data point (240).
Custom Application Calls *Invoke* function to Reset the Data Point Priority on Data Server	75	The custom application calls the *Invoke* function and resets the priority level assigned to the data point to 255, the lowest priority. At this point, all applications will be able to write to the data point.
Scheduler Updates Data Point	240	The Scheduler successfully updates the value of the data point, as the priority level used here (240) is greater than current priority assigned to the data point after being reset by the *Invoke* function (255).

4.3.8 Using the Delete Function on the Data Server

You can use the *Delete* function to delete a data point on the Data Server. To delete a data point, you provide an <Item> element of a <*driver*>_Dp_Cfg xsi:type that includes the <UCPTname> property of the data point to be deleted. If you do not specify the xsi:type, a fault may be returned stating that the

data point is still registered on its respective bus. The following code sample demonstrates how to use the *Delete* function to delete a data point on the Data Server:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <Item xsi:type="LON_Dp_Cfg">
            <UCPTname>Net/LON/iLON App/VirtFb/nvoSwitch</UCPTname>
        </Item>
    </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/LON/iLON App/VirtFb/nvoSwitch</UCPTname>
    </Item>
  </iLonItem>
</DeleteResponse>
```

Note: The xsi:types for the data points on the LONWORKS, Modbus, M-Bus, and Virtual drivers are as follows:

Driver	**xsi:type**
LONWORKS	LON_Dp_Cfg
Modbus	MOD_Dp_Cfg
M-Bus	MBS_Dp_Cfg
Virtual	Virtual_Dp_Cfg

4.4 Using the Web Binder Application

You can use the Web Binder application to create Web connections that allow direct data exchange over a TCP/IP network between a SmartServer and another host device such as a remote SmartServer, LNS Server, or a WebBinder Target Server (a Web server that can process SOAP requests such as Apache or IIS). Once you create a Web connection, the SmartServer will send a *Write* message to the target host device in the Web connection (called a WebBinder destination) each time a source data point on the local SmartServer is updated. This means that you only need to implement the *Write* function on the Data Server to create an application on the Web server that receives WebBinder updates from the SmartServer.

You can create four types of web connections: internal bindings, peer-to-peer bindings, LNS uplink bindings, and enterprise bindings.

- An internal binding is a connection between two data points on the same SmartServer. You can create internal bindings on your local SmartServer or on a remote SmartServer that you have added to the LAN. Internal bindings are useful for translating the data between two LONWORKS devices that have incompatible formats, as well as translating data between devices on different buses (LONWORKS, Modbus, and M-Bus).
- A peer-to-peer binding is a connection between two separate SmartServers. For example, you can create a peer-to-peer binding between a data on your local SmartServer to a data point on a remote SmartServer that you have added to the LAN. Peer-to-peer bindings provide an alternative solution to IP-852 connections for connecting devices over multiple networks; however, they are

much slower (40 data point updates per second) than IP-852 connections (1,000 updates per second).

- An LNS uplink binding is a connection between a SmartServer and an LNS Server. LNS uplink bindings replace the LNS uplink feature that was used in the *e*3 release for data point connections between an *i*.LON 100 *e*3 server and an LNS Server.
- An enterprise binding is a connection between a SmartServer and a Webbinder Target Server. Enterprise bindings are useful for sending a data log, an alarm log, an event scheduler log, or any user-defined file to a central enterprise system.

If you are creating peer-to-peer bindings, LNS uplink bindings, or enterprise bindings, you must first add the host device (remote SmartServer, LNS server, or Webbinder Target Server) to the LAN on which your local SmartServer resides using the SmartServer Web interface. See Chapter 3 of the *i.LON SmartServer 2.0 User's Guide* for how to add host devices to the LAN.

Note: An application that can receive a *Write* request from the SmartServer 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 use the SOAP interface to perform the following functions on a Web connection.

Function	Description
List	List the name of the Web connections that you have added to the Data Server.
Get	Retrieve the configuration of a Web connection.
Set	Create a Web connection and add it to the Data Server, or modify the configuration of an existing Web connection.
Delete	Remove a Web connection from the Data Server.

4.4.1 Using the List Function on a Web Connection

You can use the *List* function to retrieve a list of the source data points on the local SmartServer representing the Web connection that have been added to the Data Server. The *List* function takes an <iLonItem> element that has an xSelect statement querying items of a Dp_Ref type as its input, as shown in the example below. The *List* function returns an <Item> element for each source data point in the Web connections on the local SmartServer.

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="Dp_Ref"]</xSelect>
  </iLonItem>
</List>
```

Response

```
<ListResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
    <Item >
        <UCPTname>Net/LON/iLON App/Digital Input 1/nvoClsValue_1</UCPTname>
        <UCPTannotation>Dp_In_WebBinding;xsi:type=&amp;quot;Dp_Ref&amp;quot;</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
    <Item >
        <UCPTname>Net/LON/iLON App/VirtFb/nvoSwitch</UCPTname>
```

```
        <UCPTannotation>Dp_In_WebBinding;xsi:type=&amp;quot;Dp_Ref&amp;quot;</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
  </iLonItem>
</ListResponse></SOAP-ENV:Body></SOAP-ENV:Envelope>
```

4.4.2 Using the Get Function on a Web Connection

You can use the *Get* function to retrieve the configuration of any Web connection that you have added to the SmartServer's Data Server. The input parameters you supply to the function will include an <iLonItem> element that has an xSelect statement with a Dp_Ref type, and one or more <Item> elements. Each <Item> element includes a <UCPTname> property that you can use to specify the source data point in the Web connection to be returned.

Request

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="Dp_Ref"]</xSelect>
    <Item>
        <UCPTname>Net/LON/iLON App/Digital Input 1/nvoClsValue_1</UCPTname>
    </Item>
    <Item>
        <UCPTname>Net/LON/iLON App/VirtFb/nvoSwitch</UCPTname>
    </Item>
  </iLonItem>
</Get>
```

Response

```
<GetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="Dp_Ref">
        <UCPTname>Net/LON/iLON App/Digital Input 1/nvoClsValue_1</UCPTname>
        <UCPTannotation>Dp_In_WebBinding</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-17T14:29:35.666-07:00</UCPTlastUpdate>
        <UCPTdescription>Generated by WB web UI</UCPTdescription>
        <UCPTuri>Dp_Ref.htm</UCPTuri>
        <DataPoint dpType="Target" discrim="dir_in_out">
            <UCPTname>Net/LON/iLON App/Digital Output 1/nviClaValue_1</UCPTname>
            <UCPTserviceType LonFormat="UCPTserviceType">ST_WEB_ACK</UCPTserviceType>
            <UCPTservicePath>//WebService[UCPTindex=1]</UCPTservicePath>
            <UCPTpriority>255</UCPTpriority>
            <UCPTpropagate>1</UCPTpropagate>
        </DataPoint>
    </Item>
    <Item xsi:type="Dp_Ref">
        <UCPTname>Net/LON/iLON App/VirtFb/nvoSwitch</UCPTname>
        <UCPTannotation>Dp_In_WebBinding</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-19T16:41:23.338-07:00</UCPTlastUpdate>
        <UCPTdescription>Generated by WB web UI</UCPTdescription>
        <UCPTuri>Dp_Ref.htm</UCPTuri>
        <DataPoint dpType="Target" discrim="dir_in_out">
            <UCPTname>Net/LON/iLON App/Digital Output 1/nviClaValue_1</UCPTname>
            <UCPTserviceType LonFormat="UCPTserviceType">ST_WEB_ACK</UCPTserviceType>
            <UCPTservicePath></UCPTservicePath>
            <UCPTpriority>255</UCPTpriority>
            <UCPTpropagate>1</UCPTpropagate>
        </DataPoint>
    </Item>
  </iLonItem>
</Get>
```

The *Get* function returns an <Item> element for each source data point in a Web connection referenced in the input parameters you supplied to the function. The properties included within each <Item> element are initially defined when the Web connection is added to the DataServer. You can write to these properties with the *Set* function. The following table describes these properties.

Property	Description
<UCPTname>	The name of the source data point in the Web connection in following format: <*network*/*channel*/*device*/*functional block/data point*>.
<UCPTannotation>	The direction of the data point in the Web connection. This is always Dp_In_WebBinding.
<UCPThidden>	A flag indicating whether the source data point in the Web connection is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **0** – shown **1** – hidden
<UCPTlastUpdate>	A timestamp indicating the last time the configuration of the data point was updated. This timestamp uses the ISO 8601 format, which is as follows: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Data Point was last updated. The second segment (after the **T):** HH:MM:SS.sss represents the time of day the configuration of the Data Point 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 be expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. It can be left out. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002. If it is not UTC, time shift has to be defined: 2008-02-28T09:59:53.660+01:00
<UCPTdescription>	A user-defined description of the Web connection. This can be a maximum of 201 characters long. By default, this property is "Generated by WB web UI"
<UCPTuri>	The name of the file containing the Web connection data on the SmartServer flash disk. This property is always Dp_Ref.htm.
<DataPoint> Target	The target data point in the Web connection. This data point is updated every time a *Write* function is performed on the source data point in the Web connection. This data point has the following properties: • <UCPTname>. The name of the target data point in the Web connection in following format: <*network*/*channel*/*device*/*functional block/data point*>. • <UCPTserviceType>. Web connections always use

Property	Description
	Acknowledged messaging service (ST_WEB_ACK). This means that the sending device expects to receive confirmation from the receiving device that a data point update was delivered. The sending application is notified when an update fails, but it is up to the developer of the sending device to handle the notification in the device application. If you create a Web connection with the *Set* function, this property is optional. • <UCPTservicePath>. A reference to the Web service path where SOAP calls are pushed in the following format: //WebService[UCPTindex=*x*]. For example, if you create an internal binding, x is set to 0, referring to the local SmartServer. For the first host device you add to the LAN, x is set to 1; for the second host device it is set to 2; and so on. • <UCPTpriority>. The priority level currently assigned to the Web connection for writing updated values to the target data point. This value may range from 0 to 255 (highest to lowest priority). The default priority is **255**. You can assign the Web connection a higher priority for updating the target data point. The priority you specify must be equal to or higher than the priority used by the last application that updated the data point. • <UCPTpropagate>. A flag indicating whether updates to the source data point in a Web connection are to be propagated over the LONWORKS network to the target data point. If you assign the default value 1 to this property, the change you make to the source data point will be propagated to the LONWORKS network. If you assign value 0 to this property, the change will be made in the Data Server, but it will not be propagated over the network.

4.4.3 Using the Set Function on a Web Connection

Use the *Set* function to overwrite the configuration of a Web connection, or to create a new Web connection and add it to the Data Server. The input parameters you supply to the function will include one or more <Item> elements with a Dp_Ref type. Each <Item> element specifies the <UCPTname> property of the source data point in the Web connection to be created and a <DataPoint> property referencing the target data point in the connection. The <DataPoint> property must specify the <UCPTname> of the target data point and the <UCPTservicePath> of the WebBinder destination. The <DataPoint> property may specify the <UCPT priority> and <UCPTpropagate> properties, which by default are set to 255 and 1, respectively.

You can modify or create multiple Web connections with a single *Set* message by specifying two or more target data points in their respective <DataPoint> properties. However, you should not attempt to create or write to more than 100 Web connections with a single call to the *Set* function.

The following example demonstrates how to create two peer-to-peer bindings between two SmartServers. When a *Write* function is performed on the source data point in the Web connection (Net/LON/iLON App/Digital Input 1/nvoClsValue_1), the updated value is propagated to the target data points (Net/LON/iLON App/Digital Output 1/nviClaValue_1 and Net/LON/iLON App/Digital Output 2/nviClaValue_2) in the Web connection. The target data points are stored on the Data Server of a remote SmartServer that has been added to the LAN.

Request

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="Dp_Ref">
        <UCPTname>Net/LON/iLON App/Digital Input 1/nvoClsValue_1</UCPTname>
        <DataPoint dpType="Target">
            <UCPTname>Net/LON/iLON App/Digital Output 1/nviClaValue_1</UCPTname>
            <UCPTservicePath>//WebService[UCPTindex=1]</UCPTservicePath>
            <UCPTpriority>240</UCPTpriority>
            <UCPTpropagate>1</UCPTpropagate>
        </DataPoint>
        <DataPoint dpType="Target">
            <UCPTname>Net/LON/iLON App/Digital Output 2/nviClaValue_2</UCPTname>
            <UCPTservicePath>//WebService[UCPTindex=1]</UCPTservicePath>
            <UCPTpriority>240</UCPTpriority>
            <UCPTpropagate>1</UCPTpropagate>
    </DataPoint>
    </Item>
  </iLonItem>
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/"><iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/LON/iLON App/Digital Input 1/nvoClsValue_1</UCPTname>
    </Item>
</iLonItem>
</SetResponse></SOAP-ENV:Body></SOAP-ENV:Envelope>
```

4.4.4 Using the Delete Function on a Web Connection

You can use the *Delete* function to delete a Web connection on the Data Server. To delete a Web connection, you provide an <Item> element with a Dp_Ref type and the <UCPTname> property of the source data point in the Web connection. Note that using the *Delete* function will remove all Web connections in which the specified source data point is a member. To delete a single Web connection where the source data point is a member of multiple Web connections, use the *Set* function and omit the target data point of the Web connection to be removed. The following code sample demonstrates how to use the *Delete* function to delete a Web connection:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="Dp_Ref">
        <UCPTname>Net/LON/iLON App/Digital Input 1/nvoClsValue_1</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname> Net/LON/iLON App/Digital Input 1/nvoClsValue_1</UCPTname>
    </Item>
  </iLonItem>
</DeleteResponse>
```

5 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 SmartServer supports up to ten Data Loggers. The log files for each Data Logger are stored in the location specified by the Data Logger's <UCPTlogFileName> property.

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 optionally store the ASCII-text files in compressed format to save flash memory on the SmartServer.

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 threshold level for each Data Logger. The threshold level represents a percentage. When the Data Logger's log file consumes this percentage of the memory space allocated to it, the Data Logger 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 nvoDlLevAlarm[*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 7 of this document.

You can access the data in a log file by manually opening the log file, or by using the *Read* function. You can clear data from a log using the *Clear* function, or by sending an update to the data point 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.

5.1 Overview of the Data Logger XML File

The #8000010128000000[4].UFPTdataLogger.xml file stores the configurations of each Data Logger that you have added to the SmartServer. Each Data Logger is signified by an <Item> element in the XML file. The configuration properties contained in each <Item> element define the configuration of a Data Logger, and are described later in this chapter.

You can create new Data Loggers using the *Set* function, or by manually editing the #8000010128000000[4].UFPTdataLogger.xml file. You can create up to 10 Data Loggers per SmartServer. You can add more than 10 Data Loggers if you load the dynamic v40 XIF on your SmartServer and you operate your SmartServer in Standalone mode. Note that using the v40 XIF with the SmartServer operating in LNS mode (**LNS Auto** or **LNS Manual**) is not supported.

The following represents a sample #8000010128000000[4].UFPTdataLogger.xml file for a SmartServer with two Data Loggers defined on it:

```
<?xml version="1.0" encoding="utf-8" ?>
<iLonItem xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
```

```
    <Item xsi:type="UFPTdataLogger_Cfg" >
            <UCPTname>Net/LON/iLON App/Data Logger[0]</UCPTname>
            <UCPTannotation>8000010128000000[4].UFPTdataLogger</UCPTannotation>
            <UCPThidden>0</UCPThidden>
            <UCPTlastUpdate>2008-02-27T13:41:25.910-08:00</UCPTlastUpdate>
            <UCPTuri>#8000010128000000[4].UFPTdataLogger_Cfg.htm</UCPTuri>
            <DataPoint dpType="nviEnable" discrim="dir_in" >
                    <UCPTname>Net/LON/iLON App/Data Logger[0]/nviDlEnable[0]</UCPTname>
            </DataPoint>
            <DataPoint dpType="nviClear" discrim="dir_in" >
                    <UCPTname>Net/LON/iLON App/Data Logger[0]/nviDlClear[0]</UCPTname>
            </DataPoint>
            <DataPoint dpType="nvoLevelAlarm" discrim="dir_out" >
                    <UCPTname>Net/LON/iLON App/Data Logger[0]/nvoDlLevAlarm[0]
                    </UCPTname>
            </DataPoint>
            <DataPoint dpType="nvoStatus" discrim="dir_out" >
                    <UCPTname>Net/LON/iLON App/Data Logger[0]/nvoDlStatus[0]</UCPTname>
            </DataPoint>
            <DataPoint xsi:type="UFPTdataLogger_DpRef" dpType="Input" discrim="dir_in">
                    <UCPTname>Net/LON/iLON App/Digital Input 1/nvoClsValue_1</UCPTname>
                    <UCPTlogMinDeltaTime>900</UCPTlogMinDeltaTime>
                    <UCPTpollRate>900</UCPTpollRate>
            </DataPoint>
            <UCPTlogType LonFormat="UCPTlogType">LT_HISTORICAL</UCPTlogType>
            <UCPTlogSize>100</UCPTlogSize>
            <UCPTlogFormat LonFormat="UCPTlogFormat">LF_TEXT</UCPTlogFormat>
            <UCPTlogLevelAlarm>50</UCPTlogLevelAlarm>
            <UCPTlogFileName>Net/LON/iLON App/Data Logger[0].csv</UCPTlogFileName>
    </Item>
    <Item xsi:type="UFPTdataLogger_Cfg" >
            <UCPTname>Net/LON/iLON App/Data Logger[1]</UCPTname>
            <UCPTannotation>8000010128000000[4].UFPTdataLogger</UCPTannotation>
            <UCPThidden>0</UCPThidden>
            <UCPTlastUpdate>2008-02-27T13:41:57.430-08:00</UCPTlastUpdate>
            <UCPTuri>#8000010128000000[4].UFPTdataLogger_Cfg.htm</UCPTuri>
            <DataPoint dpType="nviEnable" discrim="dir_in" >
                    <UCPTname>Net/LON/iLON App/Data Logger[1]/nviDlEnable[1]</UCPTname>
            </DataPoint>
            <DataPoint dpType="nviClear" discrim="dir_in" >
                    <UCPTname>Net/LON/iLON App/Data Logger[1]/nviDlClear[1]</UCPTname>
            </DataPoint>
            <DataPoint dpType="nvoLevelAlarm" discrim="dir_out" >
                    <UCPTname>Net/LON/iLON App/Data Logger[1]/nvoDlLevAlarm[1]
                    </UCPTname>
            </DataPoint>
            <DataPoint dpType="nvoStatus" discrim="dir_out" >
                    <UCPTname>Net/LON/iLON App/Data Logger[1]/nvoDlStatus[1]</UCPTname>
            </DataPoint>
            <DataPoint xsi:type="UFPTdataLogger_DpRef" dpType="Input" discrim="dir_in">
                    <UCPTname>Net/LON/iLON App/Digital Input 2/nvoClsValue_2</UCPTname>
                    <UCPTlogMinDeltaTime>900</UCPTlogMinDeltaTime>
                    <UCPTpollRate>900</UCPTpollRate>
            </DataPoint>
            <UCPTlogType LonFormat="UCPTlogType">LT_HISTORICAL</UCPTlogType>
            <UCPTlogSize>100</UCPTlogSize>
            <UCPTlogFormat LonFormat="UCPTlogFormat">LF_TEXT</UCPTlogFormat>
            <UCPTlogLevelAlarm>50</UCPTlogLevelAlarm>
            <UCPTlogFileName>Net/LON/iLON App/Data Logger[1].csv</UCPTlogFileName>
    </Item>
</iLonItem>
```

5.2 Creating and Modifying the Data Logger XML File

You can create and modify the #8000010128000000[4].UFPTdataLogger.xml file with the *Set* function. The following section, *Data Logger SOAP Interface*, describes how to use the *Set* function and the other SOAP functions provided for the Data Logger application.

Alternatively, you can create and modify the #8000010128000000[4].UFPTdataLogger.xml file manually and download it to the SmartServer via FTP. Echelon does not recommend this, as the

SmartServer will require a reboot to read the configuration of the downloaded file. Additionally, the SmartServer 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 #8000010128000000[4].UFPTdataLogger.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.

5.3 Data Logger SOAP Interface

You can use the SOAP interface to perform the following functions on a Data Logger application:

Function	Description
List	Generate a list of the Data Loggers that you have added to the SmartServer.
Get	Retrieve the configuration of any Data Logger that you have added to the SmartServer.
Set	Create a new Data Logger, or overwrite the configuration of an existing Data Logger.
Read	Read a portion or all of the entries stored in a Data Logger log file.
Clear	Remove a portion or all of the log entries stored in a Data Logger log file.
Delete	Delete a Data Logger.

Note: Section 21.1.2, *Creating and Reading a Data Logger in Visual C# NET*, includes a C# programming example demonstrating how to use the Data Logger SOAP interface to create and read a data logger. Section 21.2.2, *Creating and Reading a Data Logger in Visual Basic. NET*, includes a Visual Basic example demonstrating how to do this.

Section 22.3.2, *Creating and Reading a Data Logger in Java*, includes a Java example demonstrating how to create and read a data logger.

5.3.1 Using the List Function on a Data Logger

Use the *List* function to retrieve a list of the Data Loggers that you have added to the SmartServer. The List function takes an <iLonItem> element that includes an xSelect statement querying items of a UFPTdataLogger_Cfg type as its input, as shown in the example below. The *List* function returns an <Item> element for each Data Logger that you have added to the SmartServer. The next section describes the properties included in each of these elements.

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

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
    <xSelect>//Item[@xsi:type="UFPTdataLogger_Cfg"]</xSelect>
    </iLonItem>
</List>
```

Response

```
<ListResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
```

```
  <iLonItem>
    <Item>
        <UCPTname>Net/LON/iLON App/Data Logger[0]</UCPTname>
        <UCPTannotation>#8000010128000000[4].UFPTdataLogger;xsi:type="LON_Fb_Cfg"
        </UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTitemStatus LonFormat="UCPTitemStatus">IS_NOTSYNCED</UCPTitemStatus>
    </Item>
  </iLonItem>
</ListResponse>
```

5.3.2 Using the Get Function on a Data Logger

You can use the *Get* function to retrieve the configuration of any Data Logger that you have added to the SmartServer. You must reference the Data Logger whose configuration is to be returned by its <UCPTname> in the input you supply to the function, as shown in the example below.

Request

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
        <iLonItem>
                <Item xsi:type="UFPTdataLogger_Cfg">
                        <UCPTname>Net/LON/iLON App/Data Logger[0]</UCPTname>
                </Item>
        </iLonItem>
</Get>
```

Response

```
<GetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTdataLogger_Cfg" >
        <UCPTname>Net/LON/iLON App/Data Logger[0]</UCPTname>
        <UCPTannotation>8000010128000000[4].UFPTdataLogger</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-02-28T11:54:06.890-08:00</UCPTlastUpdate>
        <UCPTuri>#8000010128000000[4].UFPTdataLogger_Cfg.htm</UCPTuri>
        <DataPoint dpType="nviEnable" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Data Logger[0]/nviDlEnable[0]</UCPTname>
        </DataPoint>
        <DataPoint dpType="nviClear" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Data Logger[0]/nviDlClear[0]</UCPTname>
        </DataPoint>
        <DataPoint dpType="nvoLevelAlarm" discrim="dir_out" >
            <UCPTname>Net/LON/iLON App/Data Logger[0]/nvoDlLevAlarm[0]</UCPTname>
        </DataPoint>
        <DataPoint dpType="nvoStatus" discrim="dir_out" >
            <UCPTname>Net/LON/iLON App/Data Logger[0]/nvoDlStatus[0]</UCPTname>
        </DataPoint>
        <DataPoint xsi:type="UFPTdataLogger_DpRef" dpType="Input" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Digital Input 1/nviClsValueFb_1</UCPTname>
            <UCPTlogMinDeltaTime>900</UCPTlogMinDeltaTime>
            <UCPTpollRate>900</UCPTpollRate>
        </DataPoint>
        <UCPTlogType LonFormat="UCPTlogType">LT_HISTORICAL</UCPTlogType>
        <UCPTlogSize>100</UCPTlogSize>
        <UCPTlogFormat LonFormat="UCPTlogFormat">LF_TEXT</UCPTlogFormat>
        <UCPTlogLevelAlarm>50</UCPTlogLevelAlarm>
        <UCPTlogFileName>Building/LON/iLON App/Data Logger[0].csv</UCPTlogFileName>
    </Item>
  </iLonItem>
</GetResponse>
```

The function returns an <Item> element for each Data Logger referenced in the input parameters 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 *Set* function. The following table describes these properties.

Property	Description
<UCPTname>	The name of the data logger in the following format: <*network*/*channel*/*device*/*functional block*>.
<UCPTannotation>	The program ID and functional profile template used by the data logger. This property is always 8000010128000000[4].UFPTdataLogger.
<UCPThidden>	A flag indicating whether the data logger functional block is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **0** – shown **1** – hidden
<UCPTitemStatus>	This property only appears if the data logger is not synchronized with an LNS network database or it has been deleted. In this case, it has the following values: **IS_NOTSYNCED** **IS_DELETED**
<UCPTlastUpdate>	A timestamp indicating the last time the configuration of the Data Logger was updated. This timestamp uses the format ISO 8601: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm 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 (after the **T):** HH:MM:SS.sss 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 be expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. It can be left out. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002. If it is not UTC, time shift has to be defined: 2008-02-28T09:59:53.660+01:00.
<UCPTuri>	The name of the file containing the configuration web page for the Data Logger on the SmartServer flash disk, absolute or relative to /web/user/echelon. This property is #8000010128000000[4].UFPTdataLogger_Cfg.htm by default.

Property	Description
<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. The total size of the log files for all Data Loggers (and Alarm Notifiers) on the SmartServer can not exceed the size of the flash memory stored in the SmartServer. The SmartServer will stop writing to the log files when it only has 256 Kb of flash memory remaining.
<UCPTlogFormat>	Either LF_TEXT, LF_BINARY or LF_COMPRESSED. This property indicates whether the log file the Data Logger creates will be an ASCII-text formatted .csv file (LF_TEXT), a proprietary binary format (LF_BINARY), or an ASCII-text file in compressed format (.gz file extension) (LF_COMPRESSED). You can use the LF_COMPRESSED format to save flash memory space on the SmartServer. All you need to do is extract the .csv file from the .gz file to view the log file. You can extract the file with the **decompress** console command, as described in Appendix B of the *i.LON SmartSever User's Guide.*
<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 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 7, *Alarm Notifier*. You can determine the current log level of a Data Logger with the *Read* function. You could also use the *Read* function to read the value field of the 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 with the *Clear* function, or by updating the value assigned to the nviDlClear[X] data point, 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).

Property	Description
<UCPTlogFileName>	The path of the data log file on the SmartServer flash disk, relative to the /root/data folder and including the extension of the format. By default, a data log file is stored in the root/data/Net/LON/i.LON App (Internal) folder, and it is named Data Logger [*x*], where *x* is the index number of the Data Logger functional block.
<DataPoint>	A Data Logger can record updates for multiple data points. The data points for which the Data Logger will record updates are defined by a list of <DataPoint> elements that have an xsi type attribute of "UFPTdataLogger_DpRef " and a dpType attribute of "Input". 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 data point reference that determine when an update to that data point will be logged. See the following table for descriptions of these data point properties.

The data points monitored by a Data Logger are defined by a list of <DataPoint> elements that have xsi type ="UFPTdataLogger_DpRef " and dpType="Input" attributes. The following table describes the properties that should be defined within each data point reference.

Property	Description
<UCPTname>	The name of the data point to be monitored by the Data Logger in the following format: *<network/channel/device/functional block/data point>*.
<UCPTlogMinDeltaTime>	The minimum amount of time that must pass between log entries for the data point, in seconds. 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.
<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 *Get* 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

Property	Description
	214,748,364.0 seconds. The Data Logger will check for updates to the data point at this interval. If you do not want to poll data before updates to the log are possible, you should set this to a value greater than or equal to the value specified for the <*UCPTlogMinDeltaTime*> property. If you use the default poll rate of 0 seconds, the Data Logger will record each update 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 by the update. You should note that other SmartServer 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 a Data Logger is polling the same 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 choose the greatest common divisor and therefore 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.

5.3.3 Using the Set Function on a Data Logger

Use the *Set* function to create new Data Loggers, or to overwrite the configuration of existing Data Loggers. The Data Loggers to be created or written are signified by a list of <Item> elements in the input parameters supplied to the function. The properties you must define within each <Item> element are the same, whether you are creating a new Data Logger or modifying an existing Data Logger. The previous section, *Using the Get Function on a Data Logger*, describes these properties.

Note: If you specify a data logger with the <UCPTname> element, the *Set* function deletes the specified data logger before the specified parameters are set. If the <UCPTname> element is not specified, a new data logger is created.

When modifying an existing Data Logger, any optional properties left out of the input will be erased. Old values will not be preserved, 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 *Set* function will generate the #8000010128000000[4].UFPTdataLogger.xml file in the root/config/network/<network>/<channel>/iLONApp ||<device> directory of the SmartServer, if the file does not already exist.

When creating or modifying a Data Logger with the *Set* function, you may want to use output from the *Get* function as the basis for your input. 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.

Request

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <Item xsi:type="LON_Fb_Cfg">
            <UCPTname>Net/LON/iLON App/Data Logger[3]</UCPTname>
            <UCPTannotation>#8000010128000000[4].UFPTdataLogger;xsi:type="LON_Fb_Cfg"
            </UCPTannotation>
            <UCPThidden>0</UCPThidden>
            <UCPTlastUpdate>2008-02-28T11:53:57.930-08:00</UCPTlastUpdate>
            <UCPTuri>LON_Fb_Cfg.htm</UCPTuri>
                <DataPoint dpType="nviClear" discrim="dir_in">
                    <UCPTname>Net/LON/iLON App/Data Logger[3]/nviDlClear[3]</UCPTname>
                    <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
                </DataPoint>
                <DataPoint dpType="nviEnable" discrim="dir_in">
                    <UCPTname>Net/LON/iLON App/Data Logger[3]/nviDlEnable[3]</UCPTname>
                    <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
                </DataPoint>
                <DataPoint dpType="nvoStatus" discrim="dir_out">
                    <UCPTname>Net/LON/iLON App/Data Logger[3]/nvoDlStatus[3]</UCPTname>
                    <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
                </DataPoint>
                <DataPoint dpType="nvoLevelAlarm" discrim="dir_out">
                    <UCPTname>Net/LON/iLON App/Data Logger[3]/nvoDlLevAlarm[3]</UCPTname>
                    <UCPTformatDescription>#0000000000000000[0].SNVT_alarm</UCPTformatDescription>
                </DataPoint>
            <UCPTfbIndex xsi:type="number">9</UCPTfbIndex>
        <UCPTfptKey>#8000010128000000[4].UFPTdataLogger</UCPTfptKey>
        <UCPTdynamic xsi:type="string" LonFormat="UCPTdynamic">DDT_STATIC</UCPTdynamic>
    </Item>
 </iLonItem>
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <UCPTfaultCount>0</UCPTfaultCount>
        <Item>
            <UCPTname>Net/LON/iLON App/Data Logger[3]</UCPTname>
        </Item>
    </iLonItem>
</SetResponse>
```

5.3.4 Using the Read Function on a Data Logger

Use the *Read* function to retrieve entries from the log files generated by your Data Loggers. You can specify which log entries the function will return by filling the properties described in the following table into the input you supply to the function.

You could use a *Read* request to generate a list of updates recorded for a specific data point or to generate a list of data point updates recorded during a specific interval.

Note: You should not attempt to read more than 150 log entries with a single *Read* request. You can use the following position() expression to ensures that a maximum of 64 values is returned: “[position()> =last()-64]“

Request (updates recorded for a specific data point)

```
<Read xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <xSelect> //Item[@xsi:type="UFPTdataLogger_Data"]
                        [UCPTpointName="Net/LON/iLON App/Digital Output 1/nviClaValue_1"]
        </xSelect>
   </iLonItem>
</Read>
```

Request (data point updates recorded during a specific interval)

```
<Read xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <xSelect>//Item [UCPTlastUpdate&gt;=" 2008-02-28T13:15:00.000+00:00"]
                        [UCPTlastUpdate&lt;=" 2008-02-28T13:20:00.360+00:00"]
                         [position()&gt; =last()-64][@xsi:type="UFPTdataLogger_Data"]
        </xSelect>
        <Item>
            <UCPTname>Net/LON/iLON App/Data Logger[3]</UCPTname>
        </Item>
    </iLonItem>
</Read>
```

Response

```
<ReadResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="UFPTdataLogger_Meta_Data">
        <UCPTname>Net/LON/iLON App/Data Logger[3]</UCPTname>
        <UCPTlastUpdate>2008-02-28T13:30:10.010-08:00</UCPTlastUpdate>
        <UCPTstart>2008-02-28T13:02:07.220-08:00</UCPTstart>
        <UCPTstop>2008-02-28T13:30:10.000-08:00</UCPTstop>
        <UCPTmodificationNumber>0</UCPTmodificationNumber>
        <UCPTlogLevel>13.392</UCPTlogLevel>
        <UCPTtotalCount>90</UCPTtotalCount>
    </Item>
    <Item xsi:type="UFPTdataLogger_Data" >
        <UCPTname>Net/LON/iLON App/Digital Output 1/nviClaValue_1</UCPTname>
        <UCPTaliasName>nviClaValue_1</UCPTaliasName>
        <UCPTlastUpdate>2008-02-28T13:18:00.220-08:00</UCPTlastUpdate>
        <UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch">0.0 0</UCPTvalue>
        <UCPTvalue LonFormat="UCPTvalueDef">OFF</UCPTvalue>
        <UCPTpointStatus LonFormat="UCPTpointStatus">AL_NO_CONDITION</UCPTpointStatus>
        <UCPTpriority>255</UCPTpriority>
        <UCPTmetaDataPath>//*[@xsi:type="UFPTdataLogger_Meta_Data"]
                           [UCPTname="Net/LON/iLON App/Data Logger[3]"]
        </UCPTmetaDataPath>
    </Item>
    <Item xsi:type="UFPTdataLogger_Data" >
        <UCPTname>Net/LON/iLON App/Digital Output 1/nviClaValue_1</UCPTname>
        <UCPTaliasName>nviClaValue_1</UCPTaliasName>
        <UCPTlastUpdate>2008-02-28T13:19:00.060-08:00</UCPTlastUpdate>
        <UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch">100.0 1</UCPTvalue>
        <UCPTvalue LonFormat="UCPTvalueDef">ON</UCPTvalue>
        <UCPTpointStatus LonFormat="UCPTpointStatus">AL_NO_CONDITION</UCPTpointStatus>
        <UCPTpriority>255</UCPTpriority>
        <UCPTmetaDataPath>//*[@xsi:type="UFPTdataLogger_Meta_Data"]
                           [UCPTname="Net/LON/iLON App/Data Logger[3]"]
        </UCPTmetaDataPath>
    </Item>
 </iLonItem>
</ReadResponse></SOAP-ENV:Body></SOAP-ENV:Envelope>
```

In addition to the requested log entries, the *Read* function returns an <Item> of type "UFPTdataLogger_Meta_Data" for each log file from which entries were read. This item has the following properties

<UCPTname>	The name of the data logger from which entries were read in the following format: <*network*/*channel*/*device*/*functional block*>.
<UCPTlastUpdate>	A timestamp indicating the time that the last log entry was made.
<UCPTstart> <UCPTstop>	Timestamps indicating the log times of the first and last log

	entries in the log file.
<UCPTmodificationNumber>	A counter indicating the number of times the log file has been modified. The counter is not increased when data is added to the end of the log, but only if some modifications are made to the existing data.
<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 *Read* function returns an <Item> element describing each log entry that met the selection criteria you defined in the input parameters. The following table lists the properties within each of these elements.

Property	Description
<UCPTname>	The name of the data point in the following format: <*network*/*channel*/*device*/*functional block*/*data point*>.
<UCPTaliasName>	The default or user-defined nickname provided for the data point.
<UCPTlastUpdate>	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*.
<UCPTvalue>	The value the data point was updated to when the log entry was made. The value may be presented in the following two LonFormats: • LonFormat="#<*programID*>[*scope*].<*data type*>". The format is specified by the data type defined for the data point. For a **SNVT_switch** data point, this value could be 100.0 1 or 0.0 0, for example. • LonFormat="UCPTvalueDef". The value defined for the data point by a preset. For a **SNVT_switch** data point, this value could be ON or OFF, for example. If a preset is not defined for the data point, this value is AL_NUL.
<UCPTunit>	The unit type of the data point.
<UCPTpointStatus>	The status the data point was updated to when the log entry was made.
<UCPTpriority>	The priority level currently assigned to the data point (0-255). The priority level of a data point determines which applications can write to its value. You can modify the value of this property with the *Write* or *ResetPriority* functions.

5.3.4.1 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.sss]+/-[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 fractions of a second.

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. If the local time matches UTC, the third segment will be replaced by the letter Z.

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.

5.3.5 Using the Clear Function on a Data Logger

You can use the *Clear* 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 using xSelect statements. If no filter is specified with an xSelect statement, the whole data log will be deleted.

Note: This function only deletes the log entries. You can delete the Data Logger itself using the *Delete* function.

The following call to the *Clear* function deletes up to 100 log entries from a data logger that occurred after the specified date and time.

Request

```
<Clear xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <xSelect>
            //Item[UCPTlastUpdate&gt;="2008-02-28T14:00:00.000-8:00"]
                  [position()&gt;=0 and position()&lt;=99]
        </xSelect>
        <Item>
            <UCPTname>Net/LON/iLON App/Data Logger[0]</UCPTname>
        </Item>
    </iLonItem>
</Clear>
```

Response

```
<ClearResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem >

        <UCPTfaultCount>0</UCPTfaultCount>
        <Item xsi:type="UFPTdataLogger_ClearResponse" >
            <UCPTname>Net/LON/iLON App/Data Logger[0]</UCPTname>
            <UCPTlastUpdate>2008-02-28T14:14:08.340-08:00</UCPTlastUpdate>
            <UCPTstart>2008-02-28T12:28:52.780-08:00</UCPTstart>
```

```
            <UCPTstop>2008-02-28T14:00:00.070-08:00</UCPTstop>
            <UCPTmodificationNumber>1</UCPTmodificationNumber>
            <UCPTlogLevel>1.485</UCPTlogLevel>
            <UCPTtotalCount>8</UCPTtotalCount>
        </Item>
    </iLonItem>
</ClearResponse></SOAP-ENV:Body></SOAP-ENV:Envelope>
```

The *Clear* function returns the following information related to the log file from which entries were deleted:

Property	Description
<UCPTname>	The name of the data logger in which entries were cleared in the following format: <*network*/*channel*/*device*/*functional block*>.
<UCPTlastUpdate>	A timestamp indicating the time that the log was cleared.
<UCPTfileName>	The name of the log file the Data Logger is using.
<UCPTstart> <UCPTstop>	Timestamps indicating the times of the first and last log entries in the log file.
<UCPTmodificationNumber>	A counter indicating the number of times the log file has been modified. The counter is not increased when data is added to the end of the log, but only if some modifications are made to the existing data.
<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	The total number of entries contained in the data log read by the function.

5.3.6 Using the Delete Function on a Data Logger

You can use the *Delete* function to delete a Data Logger. To delete a Data Logger, you provide an <Item> element with a UFPTdataLogger_Cfg type that includes the <UCPTname> property of the data logger to be deleted. The following code sample demonstrates how to use the *Delete* function to delete a Data Logger:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <Item xsi:type="UFPTdataLogger_Cfg">
            <UCPTname>Net/LON/iLON App/Data Logger[2]</UCPTname>
        </Item>
    </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/LON/iLON App/Data Logger[2]</UCPTname>
    </Item>
  </iLonItem>
</DeleteResponse>
```

6 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 specify 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 **SNVT_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 7, *Alarm Notifier*.

6.1 Overview of the Alarm Generator XML File

The #8000010128000000[4].UFPTalarmGenerator.xml file stores the configuration of the Alarm Generators that you have added to the SmartServer. Each Alarm Generator is signified by an <Item> element in the XML file.

You can create new Alarm Generators using the *Set* function, or by manually editing the #8000010128000000[4].UFPTalarmGenerator.xml file, and rebooting the SmartServer. You can create up to 40 Alarm Generators per SmartServer. You can add more than 40 Alarm Generators if you load the dynamic v40 XIF on your SmartServer and you operate your SmartServer in Standalone mode. Note that using the v40 XIF with the SmartServer operating in LNS mode (**LNS Auto** or **LNS Manual**) is not supported.

The following represents a sample #8000010128000000[4].UFPTalarmGenerator.xml file with one Alarm Generator.

```
<Item xsi:type="UFPTalarmGenerator_Cfg" >
        <UCPTname>Net/LON/iLON App/Alarm Generator[0]</UCPTname>
        <UCPTannotation>8000010128000000[4].UFPTalarmGenerator</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-02-28T15:29:23.220-08:00</UCPTlastUpdate>
        <UCPTuri>#8000010128000000[4].UFPTalarmGenerator_Cfg.htm</UCPTuri>
        <DataPoint dpType="nviEnable" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Alarm Generator[0]/nviAgEnable[0]</UCPTname>
        </DataPoint>
        <DataPoint dpType="nvoAlarmFlag" discrim="dir_out" >
            <UCPTname>Net/LON/iLON App/Alarm Generator[0]/nvoAgAlarmFlag[0]</UCPTname>
        </DataPoint>
        <DataPoint dpType="nviLatchEnable" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Alarm Generator[0]/nviAgLatchEnbl[0]</UCPTname>
        </DataPoint>
        <DataPoint dpType="Input" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Digital Output 1/nviClaValue_1</UCPTname>
        </DataPoint>
        <DataPoint dpType="Compare" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Alarm Generator[0]/CompareDP</UCPTname>
        </DataPoint>
        <UCPTalrmIhbD>0.000000</UCPTalrmIhbD>
        <UCPTalarmPriority LonFormat="UCPTalarmPriority">PR_LEVEL_1</UCPTalarmPriority>
        <UCPTpollOnResetDelay>0.0</UCPTpollOnResetDelay>
        <UCPTpollRate>0.0</UCPTpollRate>
        <UCPTalarm2Description></UCPTalarm2Description>
        <UCPTcompFunction LonFormat="UCPTcompFunction">FN_EQ</UCPTcompFunction>
        <UCPTalarmSetTimeD>0.000000</UCPTalarmSetTimeD>
        <UCPTalarmClrTimeD>0.000000</UCPTalarmClrTimeD>
        <UCPTlowLimit1Offset LonFormat="UNVT_double_float"></UCPTlowLimit1Offset>
        <UCPTlowLimit2Offset LonFormat="UNVT_double_float"></UCPTlowLimit2Offset>
        <UCPThighLimit1Offset LonFormat="UNVT_double_float"></UCPThighLimit1Offset>
        <UCPThighLimit2Offset LonFormat="UNVT_double_float"></UCPThighLimit2Offset>
        <SCPThystHigh1 LonFormat="UNVT_double_float"></SCPThystHigh1>
        <SCPThystHigh2 LonFormat="UNVT_double_float"></SCPThystHigh2>
        <SCPThystLow1 LonFormat="UNVT_double_float"></SCPThystLow1>
        <SCPThystLow2 LonFormat="UNVT_double_float"></SCPThystLow2>
    </Item>
```

6.2 Creating and Modifying the Alarm Generator XML File

You can create and modify the #8000010128000000[4].UFPTalarmGenerator.xml file with the Set SOAP function. The following section, *Alarm Generator SOAP Interface*, describes how to use the *Set* function, and the other SOAP functions provided for the Alarm Generator application.

Alternatively, you can create and modify the #8000010128000000[4].UFPTalarmGenerator.xml file manually using an XML editor, and download the file to the SmartServer via FTP. Echelon does not recommend this, as the SmartServer will require a reboot to read the configuration of the downloaded file. Additionally, the SmartServer 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.

6.3 Alarm Generator SOAP Interface

The SOAP interface for the Alarm Generator application includes the following functions:

Function	Description
List	Generate a list of the Alarm Generators that you have added to the SmartServer.

Get	Retrieve the configuration of any Alarm Generator that you have added to the SmartServer.
Set	Create a new Alarm Generator, or overwrite the configuration of an existing Alarm Generator.
Delete	Delete an Alarm Generator.

6.3.1 Using the List Function on an Alarm Generator

Use the *List* function to retrieve a list of the Alarm Generators that you have added to the SmartServer. The List function takes an <iLonItem> element that includes an xSelect statement querying items of a UFPTalarmGenerator_Cfg type as its input, as shown in the example below. The *List* function returns an <Item> element for each Alarm Generator that you have added to the SmartServer. The next section describes the properties included in each of these elements.

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

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <xSelect>//Item[@xsi:type="UFPTalarmGenerator_Cfg"]</xSelect>
    </iLonItem>
</List>
```

Response

```
<ListResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>Net/LON/iLON App/Alarm Generator[0]</UCPTname>
        <UCPTannotation>#8000010128000000[4].UFPTalarmGenerator;xsi:type="LON_Fb_Cfg"
        </UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
  </iLonItem>
</ListResponse>
```

6.3.2 Using the Get Function on an Alarm Generator

You can use the *Get* function to retrieve the configuration of any Alarm Generator that you have added to the SmartServer. You must reference the Alarm Generator whose configuration is to be returned by its <UCPTname> in the input you supply to the function, as shown in the example below.

Request

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <Item>
            <UCPTname>Net/LON/iLON App/Alarm Generator[0]</UCPTname>
        </Item>
    </iLonItem>
</Get>
```

Response

```
<GetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="UFPTalarmGenerator_Cfg" >
        <UCPTname>Net/LON/iLON App/Alarm Generator[0]</UCPTname>
        <UCPTannotation>8000010128000000[4].UFPTalarmGenerator</UCPTannotation>
        <UCPThidden>0</UCPThidden>
```

```
        <UCPTlastUpdate>2008-02-28T15:45:26.060-08:00</UCPTlastUpdate>
        <UCPTuri>#8000010128000000[4].UFPTalarmGenerator_Cfg.htm</UCPTuri>
        <DataPoint dpType="nviEnable" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Alarm Generator[0]/nviAgEnable[0]</UCPTname>
        </DataPoint>
        <DataPoint dpType="nvoAlarmFlag" discrim="dir_out" >
            <UCPTname>Net/LON/iLON App/Alarm Generator[0]/nvoAgAlarmFlag[0]</UCPTname>
        </DataPoint>
        <DataPoint dpType="nviLatchEnable" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Alarm Generator[0]/nviAgLatchEnbl[0]</UCPTname>
        </DataPoint>
        <DataPoint dpType="Input" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Digital Output 1/nviClaValue_1</UCPTname>
        </DataPoint>
        <DataPoint dpType="Compare" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Alarm Generator[0]/CompareDP</UCPTname>
        </DataPoint>
        <DataPoint dpType="Alarm" discrim="dir_out" >
            <UCPTname>Net/LON/iLON App/Alarm Generator[0]/alarm</UCPTname>
        </DataPoint>
        <UCPTalrmIhbD>0.000000</UCPTalrmIhbD>
        <UCPTalarmPriority LonFormat="UCPTalarmPriority">PR_LEVEL_1</UCPTalarmPriority>
        <UCPTpollOnResetDelay>0.0</UCPTpollOnResetDelay>
        <UCPTpollRate>0.0</UCPTpollRate>
        <UCPTalarm2Description></UCPTalarm2Description>
        <UCPTcompFunction LonFormat="UCPTcompFunction">FN_EQ</UCPTcompFunction>
        <UCPTalarmSetTimeD>0.000000</UCPTalarmSetTimeD>
        <UCPTalarmClrTimeD>0.000000</UCPTalarmClrTimeD>
        <UCPTlowLimit1Offset LonFormat="UNVT_double_float"></UCPTlowLimit1Offset>
        <UCPTlowLimit2Offset LonFormat="UNVT_double_float"></UCPTlowLimit2Offset>
        <UCPThighLimit1Offset LonFormat="UNVT_double_float"></UCPThighLimit1Offset>
        <UCPThighLimit2Offset LonFormat="UNVT_double_float"></UCPThighLimit2Offset>
        <SCPThystHigh1 LonFormat="UNVT_double_float"></SCPThystHigh1>
        <SCPThystHigh2 LonFormat="UNVT_double_float"></SCPThystHigh2>
        <SCPThystLow1 LonFormat="UNVT_double_float"></SCPThystLow1>
        <SCPThystLow2 LonFormat="UNVT_double_float"></SCPThystLow2>
    </Item>
  </iLonItem>
</GetResponse>
```

The function returns an <Item> element for each Alarm Generator referenced in the input parameters supplied to the function. The properties included in each element are initially defined when the Alarm Generator is created. You can write to these properties with the *Set* function. The following table describes these properties.

Property	Description
`<UCPTname>`	The name of the Alarm Generator in the following format: *<network/channel/device/functional block>*.
`<UCPTannotation>`	The program ID and functional profile template used by the Alarm Generator. This property is always `8000010128000000[4].UFPTalarmGenerator`
`<UCPThidden>`	A flag indicating whether the Alarm Generator functional block is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **0** – shown **1** – hidden
`<UCPTitemStatus>`	This property only appears if the data logger is not synchronized with an LNS network database or it has been

Property	Description
	deleted. In this case, it has the following values: **IS_NOTSYNCED** **IS_DELETED**
<UCPTlastUpdate>	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**
<UCPTuri>	The name of the file containing the configuration web page for the Alarm Generator on the SmartServer flash disk, absolute or relative to /web/user/echelon folder. This property is #8000010128000000[4].UFPTalarmGenerator_Cfg.htm by default.
<UCPTalrmIhbT>	The time period for which alarm generation is to be inhibited after the application is enabled. This period must be entered in seconds, as a double precision floating point value.
<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*.
<UCPTpollOnResetDelay>	The time period to wait after enabling or starting the application before polling the value of the input data point, in seconds. 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 conditions when it receives event-driven updates to the data points. You should note that other SmartServer 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 a Data Logger is polling that data point every 10 seconds, then the Data Server will have to poll the value of the data

Property	Description
	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. The description field of the **SNVT_alarm2** data point selected for the Alarm Generator. This data point can be selected by setting the SNVT_alarm_2 output data point property. The description of the data point 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 **SNVT_alarm2** data point each time an alarm is generated.
`<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*.
`<UCPTalarmSetTime>`	Specifies the time period an alarm condition must exist before the Alarm Generator will consider it a valid alarm and generate an alarm. The time period must be entered in seconds, as a double precision floating point value.
`<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. The time period must be entered in seconds, as a double precision floating point value.

`<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

Property	Description
	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 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, see *Hysteresis Levels and Offset Limits*. **Note:** If you use the Get function to retrieve 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.

`<SCPThystHigh1>`
`<SCPThystHigh2>`
`<SCPThystLow1>`
`<SCPThystLow2>`
	When an alarm occurs based on one of the offset 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*.
	
	Note: If you use the *Get* function to retrieve 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.

`<DataPoint>`
 	
`Input`
	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

Property	Description
	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*. 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*.
`<DataPoint>` `Compare`	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. You can use a 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.

`<DataPoint>`
`SNVT_alarm Output`
`SNVT_alarm2 Output`
	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 SNVT_alarm output data point must use the format type **SNVT_alarm**. The data point chosen for the SNVT_alarm_2 output data point must use the format type **SNVT_alarm_2**.
	
	Use a **SNVT_alarm** data point if your system can handle this LonMark standard type for alarming. Use a **SNVT_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 *Read* function.
	
	The <UCPTpointSatus> of each alarm data point 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*.
	
	You can register these alarm data points with the Alarm Notifier application to generate alarm notifications and

Property	Description
	e-mail messages each time they are updated to an alarm status. For more information on this, see *Alarm Notifier*.

6.3.2.1 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 the following table. 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.

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

6.3.2.2 Comparison Functions

The following tables 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 the following table.

Identifier	Description
FN_GT	Greater than. An alarm will be generated if the input value is greater than the compare value.
FN_LT	Less than. An alarm will be generated if the input value is less than the compare value.
FN_GE	Greater than or equal. An alarm will be generated if the input value is greater than or equal to the compare value.
FN_LE	Less than or equal. An alarm will be generated if the input value is less than or equal to the compare value.
FN_EQ	Equal. An alarm will be generated if the input value is equal to the compare value.

Identifier	Description
FN_NE	Not equal. 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, see *Hysteresis Levels and Offset Limits*.

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

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. The following table lists the <UCPTcompFunction> identifiers you could use for these special comparisons. A description of how these comparisons are made follows the table.

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.

6.3.2.3 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.

The following table 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.

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. The following table describes how these levels are calculated for each of the offset limits listed above.

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. The following table describes this process in more detail.

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.

Event	Input Data Point Status	Comments
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 hysteresis level for the first alarm condition.	AL_NO_CONDITION	Updated back to normal status.

6.3.3 Using the Set Function on an Alarm Generator

Use the *Set* function to create new Alarm Generators, or to overwrite the configuration of existing Alarm Generators. The Alarm Generators to be created or written are signified by a list of <Item> elements in the input parameters supplied to the function. The properties you must define within each <Item> element are the same, whether you are creating a new Alarm Generator or modifying an existing Alarm Generator. The previous section, *Using the Get Function an Alarm Generator*, describes these properties.

Note: If you specify an Alarm Generator with the <UCPTname> element, the *Set* function deletes the specified Alarm Generator before the specified parameters are set. If the <UCPTname> element is not specified, a new Alarm Generator is created.

When modifying an existing Alarm Generator, any optional properties omitted from the *Set* Request, such as the input point, compare point, or SNVT_alarm and SNVT_alarm_2 output data points, will be erased. Old values will not be preserved, 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 *Set* function will generate the #8000010128000000[4].UFPTdataLogger.xml file in the root/config/network/<network>/<channel>/iLONApp ||<device> directory of the SmartServer, if the file does not already exist.

When creating or modifying an Alarm Generator with the *Set* function, you may want to use output from the *Get* function as the basis for your input. 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.

Request

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Fb_Cfg">
        <UCPTname>Net/LON/iLON App/Alarm Generator[0]</UCPTname>
        <UCPTannotation>#8000010128000000[4].UFPTalarmGenerator;xsi:type="LON_Fb_Cfg"
        </UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTuri>LON_Fb_Cfg.htm</UCPTuri>
        <DataPoint dpType="nviEnable" discrim="dir_in">
            <UCPTname>Net/LON/iLON App/Alarm Generator[0]/nviAgEnable[0]</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
        </DataPoint>
        <DataPoint dpType="nvoAlarmFlag" discrim="dir_out">
            <UCPTname>Net/LON/iLON App/Alarm Generator[0]/nvoAgAlarmFlag[0]</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
        </DataPoint>
        <DataPoint dpType="nviLatchEnable" discrim="dir_in">
            <UCPTname>Net/LON/iLON App/Alarm Generator[0]/nviAgLatchEnbl[0]</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
        </DataPoint>
        <UCPTalrmIhbD>0.0</UCPTalrmIhbD>
       <UCPTalarmPriority LonFormat="UCPTalarmPriority">PR_LEVEL_1</UCPTalarmPriority>
       <UCPTpollOnResetDelay>0.0</UCPTpollOnResetDelay>
```

```
      <UCPTpollRate>0.0</UCPTpollRate>
      <UCPTalarm2Description>none</UCPTalarm2Description>
      <UCPTcompFunction LonFormat="UCPTcompFunction">FN_EQ</UCPTcompFunction>
      <UCPTalarmSetTimeD>0.0</UCPTalarmSetTimeD>
      <UCPTalarmClrTimeD>0.0</UCPTalarmClrTimeD>
      <UCPTlowLimit1Offset LonFormat="UNVT_double_float">10000</UCPTlowLimit1Offset>
      <UCPTlowLimit2Offset LonFormat="UNVT_double_float">20000</UCPTlowLimit2Offset>
      <UCPThighLimit1Offset LonFormat="UNVT_double_float">10000</UCPThighLimit1Offset>
      <UCPThighLimit2Offset LonFormat="UNVT_double_float">20000</UCPThighLimit2Offset>
      <SCPThystHigh1 LonFormat="UNVT_double_float">5000.000000</SCPThystHigh1>
      <SCPThystHigh2 LonFormat="UNVT_double_float">5000.000000</SCPThystHigh2>
      <SCPThystLow1 LonFormat="UNVT_double_float">5000.000000</SCPThystLow1>
      <SCPThystLow2 LonFormat="UNVT_double_float">5000.000000</SCPThystLow2>
    </Item>
  </iLonItem
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/LON/iLON App/Alarm Generator[0]</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

6.3.4 Using the Delete Function on an Alarm Generator

You can use the *Delete* function to delete an Alarm Generator. To delete an Alarm Generator, you provide an <Item> element with a UFPTalarmGenerator_Cfg type that includes the <UCPTname> property of the alarm generator to be deleted. The following code sample demonstrates how to use the *Delete* function to delete an Alarm Generator:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTalarmGenerator_Cfg">
        <UCPTname>Net/LON/iLON App/Alarm Generator[0]</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname> Net/LON/iLON App/Alarm Generator[0]</UCPTname>
    </Item>
  </iLonItem>
</DeleteResponse>
```

7 Alarm Notifier

Use the Alarm Notifier application to log user-defined alarm conditions, 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 Notifier will consider alarm conditions are user-defined. You will define these conditions by creating active and passive alarm condition sets for the Alarm Notifier.

For each condition set you create, you will select an alarm type (active or passive) and a data point status. Each time an input data point is updated and its <UCPTpointStatus> matches the selected status, an alarm notification will occur. If it is generated based on a status assigned to an active alarm condition set, it is considered an *active alarm*. If it is generated based on a status 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 6*.

You may also recall from Chapter 5 that some data points exist in the Data Server to monitor the amount of memory that an Alarm Generator's log file has consumed. You could set up an Alarm Notifier to generate alarm notifications when a log file becomes full.

Alarm Destinations

You can create destinations for your Alarm Notifiers. These destinations determine how the Alarm Notifier will respond when an alarm occurs. 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 will be used when an active alarm notification occurs.

For each destination, 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 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 SmartServer. These files are named histlog*X*, 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 SmartServer.

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 with the SmartServer Web pages, by opening the log files via FTP, or by using the *Read* function. You can use the *Write* function to acknowledge and comment on the alarm notifications stored in the log files.

7.1 Overview of the AlarmNotifier XML File

The #8000010128000000[4].UFPTalarmNotifier.xml file stores the configuration of the Alarm Notifiers that you have added to the SmartServer. Each Alarm Notifier is signified by an <Item> element in the XML file.

You can create new Alarm Notifiers using the *Set* function, or by manually editing the #8000010128000000[4].UFPTalarmNotifier.xml file, and rebooting the SmartServer. You can create up to 40 Alarm Notifiers per SmartServer. You can add more than 40 Alarm Notifiers if you load the dynamic v40 XIF on your SmartServer and you operate your SmartServer in Standalone mode. Note that using the v40 XIF with the SmartServer operating in LNS mode (**LNS Auto** or **LNS Manual**) is not supported.

The following represents a sample #8000010128000000[4].UFPTalarmNotifier.xml file with one Alarm Notifier. This Alarm Notifier generates alarm notifications based on the status of an **nvoDlLevAlarm** data point. This **nvoDlLevAlarm** data point monitors the log level of a Data Logger. As you may recall from Chapter 4, 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 **nviDlClear** data point to 100.0 1. The update to **nviDlClear** will clear out the Data Logger's log file. In summary, the Alarm Notifier defined by the XML file below monitors the log level of an Alarm Generator, and empties the Data Logger's log file when it becomes full.

```
<Item xsi:type="UFPTalarmNotifier_Cfg" >
    <UCPTname>Net/LON/iLON App/Alarm Notifier[0]</UCPTname>
    <UCPTannotation>8000010128000000[4].UFPTalarmNotifier</UCPTannotation>
    <UCPThidden>0</UCPThidden>
    <UCPTlastUpdate>2008-02-29T11:16:53.190-08:00</UCPTlastUpdate>
    <UCPTuri>#8000010128000000[4].UFPTalarmNotifier_Cfg.htm</UCPTuri>
    <DataPoint dpType="nviEnable" discrim="dir_in" >
        <UCPTname>Net/LON/iLON App/Alarm Notifier[0]/nviAnEnable[0]</UCPTname>
    </DataPoint>
    <DataPoint xsi:type="UFPTalarmNotifier_Input_DpRef" dpType="Input" discrim="dir_in" >
        <UCPTname>Net/LON/iLON App/Data Logger[0]/nvoDlLevAlarm[0]</UCPTname>
        <AlarmFlags>
            <UCPTlogEnable>1</UCPTlogEnable>
            <UCPTinvisible>0</UCPTinvisible>
            <UCPTclearRequired>0</UCPTclearRequired>
            <UCPTackRequired>1</UCPTackRequired>
            <UCPTdisabled>0</UCPTdisabled>
            <UCPTcovEnabled>1</UCPTcovEnabled>
        </AlarmFlags>
        <UCPTalarmGroup>0</UCPTalarmGroup>
        <UCPTalarmPriority2>0</UCPTalarmPriority2>
        <UCPTdescription/>
    </DataPoint>
    <DataPoint xsi:type="UFPTalarmNotifier_DpRef" dpType="Output" discrim="dir_out" >
        <UCPTname>Net/LON/iLON App/Data Logger[0]/nviDlClear[0]</UCPTname>
        <UCPTnickName>Net/LON/iLON App/Data Logger[0]/nviDlClear[0]</UCPTnickName>
    </DataPoint>
    <SCPTdelayTime>0</SCPTdelayTime>
    <UCPTsumLogSize>50</UCPTsumLogSize>
    <UCPThistLogSize>100</UCPThistLogSize>
    <UCPTlogFormat LonFormat="UCPTlogFormat">LF_TEXT</UCPTlogFormat>
    <UCPTsumLogFileName>Net/LON/iLON App/Alarm Notifier[0]_Summary.csv</UCPTsumLogFileName>
    <UCPThistLogFileName>Net/LON/iLON App/Alarm Notifier[0]_History.csv</UCPThistLogFileName>
```

```
        <UCPTemailAggregTime>0</UCPTemailAggregTime>
        <Mail>
            <UCPTindex>0</UCPTindex>
            <UCPTnickName>Alarm Notification </UCPTnickName>
            <UCPTemailAddress>user@echelon.com</UCPTemailAddress>
            <UCPTemailFormat>{status}&#13;{new_line}{alarm_time}</UCPTemailFormat>
            <UCPTemailSubject>Data Logger at Alarm Level</UCPTemailSubject>
            <UCPTemailAttachment/>
        </Mail>
        <ActiveAlarm>
            <UCPTindex>5</UCPTindex>
            <UCPTlevel>240</UCPTlevel>
            <UCPTalarmText>Alarm</UCPTalarmText>
            <UCPTalarmCondition LonFormat="UCPTalarmCondition">AL_ALM_CONDITION</UCPTalarmCondition>
        </ActiveAlarm>
        <PassiveAlarm>
            <UCPTindex>0</UCPTindex>
            <UCPTlevel>255</UCPTlevel>
            <UCPTalarmText>Online</UCPTalarmText>
            <UCPTalarmCondition LonFormat="UCPTalarmCondition">AL_NO_CONDITION</UCPTalarmCondition>
        </PassiveAlarm>
        <AlarmDest>
            <UCPTindex>0</UCPTindex>
            <UCPTenablePath/>
            <ActiveDest>
                <UCPTmailPath>Mail[UCPTnickName="Alarm Notification "]</UCPTmailPath>
                <UCPTdataPointPath>DataPoint[@dpType="Output" and UCPTnickName="Net/LON/iLON App/Data
                Logger[0]/nviDlClear[0]"]</UCPTdataPointPath>
                <UCPTpointValue>ON</UCPTpointValue>
                <UCPTminLevel>255</UCPTminLevel>
                <UCPTmaxLevel>0</UCPTmaxLevel>
                <UCPTnackDelay>0</UCPTnackDelay>
            </ActiveDest>
            <PassiveDest>
                <UCPTminLevel>255</UCPTminLevel>
                <UCPTmaxLevel>0</UCPTmaxLevel>
                <UCPTnackDelay>0</UCPTnackDelay>
            </PassiveDest>
        </AlarmDest>
    </Item>
</iLonItem>
```

7.2 Creating and Modifying the Alarm Notifier XML File

You can create and manage the #8000010128000000[4].UFPTalarmNotifier.xml file with the *Set* SOAP function. The following section, *Alarm Notifier SOAP Interface*, describes how to use *Set* and the other SOAP functions provided for the Alarm Notifier application.

Alternatively, you can create and manage the #8000010128000000[4].UFPTalarmNotifier.xml file manually with an XML editor and download it to the SmartServer via FTP. Echelon does not recommend this, as the SmartServer will require a reboot to read the configuration of the downloaded file. Additionally, the SmartServer 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.

7.3 Alarm Notifier SOAP Interface

You can use the SOAP interface to perform the following functions on an Alarm Notifier application:

Function	Description
List	Generate a list of the Alarm Notifiers that you have added to the SmartServer.
Get	Retrieve the configuration of any Alarm Notifier that you have added to the SmartServer.
Set	Create a new Alarm Notifier, or overwrite the configuration of an existing Alarm Notifier.
Read	Read a portion or all of the entries stored in a Alarm Notifier log file.
Write	Acknowledge an alarm notification or a group of alarm notifications. You can optionally insert comments into the log entry for each alarm notification with this function.
Clear	Remove a portion or all of the log entries stored in an Alarm Notifier log file.
Delete	Delete an Alarm Notifier.

7.3.1 Using the List Function on an Alarm Notifier

Use the *List* function to retrieve a list of the Alarm Notifiers that you have added to the SmartServer. The List function takes an <iLonItem> element that includes an xSelect statement querying items of a UFPTalarmNotifier_Cfg type as its input, as shown in the example below. The *List* function returns an <Item> element for each Alarm Notifier that you have added to the SmartServer. The next section describes the properties included in each of these elements.

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

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <xSelect>//Item[@xsi:type="UFPTalarmNotifier_Cfg"]</xSelect>
    </iLonItem>
</List>
```

Response

```
<ListResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>Net/LON/iLON App/Alarm Notifier[0]</UCPTname>
        <UCPTannotation>#8000010128000000[4].UFPTalarmNotifier;xsi:type="LON_Fb_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
  </iLonItem>
</ListResponse>
```

7.3.2 Using the Get Function on an Alarm Notifier

You can use the *Get* function to retrieve the configuration of any Alarm Notifier that you have added to the SmartServer. You must reference the Alarm Notifier whose configuration is to be returned by its <UCPTname> in the input you supply to the function, as shown in the example below.

Request

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <Item>
            <UCPTname>Net/LON/iLON App/Alarm Notifier[0]</UCPTname>
        </Item>
    </iLonItem>
</Get>
```

Response

```
<GetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
  <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="UFPTalarmNotifier_Cfg" >
        <UCPTname>Net/LON/iLON App/Alarm Notifier[0]</UCPTname>
        <UCPTannotation>8000010128000000[4].UFPTalarmNotifier</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-02-29T11:16:53.190-08:00</UCPTlastUpdate>
        <UCPTuri>#8000010128000000[4].UFPTalarmNotifier_Cfg.htm</UCPTuri>
        <DataPoint dpType="nviEnable" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Alarm Notifier[0]/nviAnEnable[0]</UCPTname>
        </DataPoint>
        <DataPoint xsi:type="UFPTalarmNotifier_Input_DpRef" dpType="Input" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Data Logger[0]/nvoDlLevAlarm[0]</UCPTname>
            <AlarmFlags>
                <UCPTlogEnable>1</UCPTlogEnable>
                <UCPTinvisible>0</UCPTinvisible>
                <UCPTclearRequired>0</UCPTclearRequired>
                <UCPTackRequired>1</UCPTackRequired>
                <UCPTdisabled>0</UCPTdisabled>
                <UCPTcovEnabled>1</UCPTcovEnabled>
            </AlarmFlags>
            <UCPTalarmGroup>0</UCPTalarmGroup>
            <UCPTalarmPriority2>0</UCPTalarmPriority2>
            <UCPTdescription/>
        </DataPoint>
        <DataPoint xsi:type="UFPTalarmNotifier_DpRef" dpType="Output" discrim="dir_out" >
            <UCPTname>Net/LON/iLON App/Data Logger[0]/nviDlClear[0]</UCPTname>
            <UCPTnickName>Net/LON/iLON App/Data Logger[0]/nviDlClear[0]</UCPTnickName>
        </DataPoint>
        <SCPTdelayTime>0</SCPTdelayTime>
        <UCPTsumLogSize>50</UCPTsumLogSize>
        <UCPThistLogSize>100</UCPThistLogSize>
        <UCPTlogFormat LonFormat="UCPTlogFormat">LF_TEXT</UCPTlogFormat>
        <UCPTsumLogFileName>Net/LON/iLON App/Alarm Notifier[0]_Summary.csv</UCPTsumLogFileName>
        <UCPThistLogFileName>Net/LON/iLON App/Alarm Notifier[0]_History.csv</UCPThistLogFileName>
        <UCPTemailAggregTime>0</UCPTemailAggregTime>
        <Mail>
            <UCPTindex>0</UCPTindex>
            <UCPTnickName>Alarm Notification </UCPTnickName>
            <UCPTemailAddress>jduval@echelon.com</UCPTemailAddress>
            <UCPTemailFormat>{status}{alarm_time}</UCPTemailFormat>
            <UCPTemailSubject>Data Logger at Alarm Level</UCPTemailSubject>
            <UCPTemailAttachment/>
        </Mail>
        <ActiveAlarm>
            <UCPTindex>5</UCPTindex>
            <UCPTlevel>240</UCPTlevel>
            <UCPTalarmText>Alarm</UCPTalarmText>
            <UCPTalarmCondition LonFormat="UCPTalarmCondition">AL_ALM_CONDITION</UCPTalarmCondition>
        </ActiveAlarm>
        <PassiveAlarm>
            <UCPTindex>0</UCPTindex>
```

```
            <UCPTlevel>255</UCPTlevel>
            <UCPTalarmText>Online</UCPTalarmText>
            <UCPTalarmCondition LonFormat="UCPTalarmCondition">AL_NO_CONDITION</UCPTalarmCondition>
        </PassiveAlarm>
        <AlarmDest>
            <UCPTindex>0</UCPTindex>
            <UCPTenablePath/>
            <ActiveDest>
                <UCPTmailPath>Mail[UCPTnickName="Alarm Notification "]</UCPTmailPath>
                <UCPTdataPointPath>DataPoint[@dpType="Output" and UCPTnickName="Net/LON/iLON App/
                  Data Logger[0]/nviDlClear[0]"]
                </UCPTdataPointPath>
                <UCPTpointValue>ON</UCPTpointValue>
                <UCPTminLevel>255</UCPTminLevel>
                <UCPTmaxLevel>0</UCPTmaxLevel>
                <UCPTnackDelay>0</UCPTnackDelay>
            </ActiveDest>
            <PassiveDest>
                <UCPTminLevel>255</UCPTminLevel>
                <UCPTmaxLevel>0</UCPTmaxLevel>
                <UCPTnackDelay>0</UCPTnackDelay>
            </PassiveDest>
        </AlarmDest>
    </Item>
  </iLonItem>
</GetResponse>
```

The function returns an <Item> element for each Alarm Notifier referenced in the input parameters supplied to the function. The properties included in each element are initially defined when the Alarm Notifier is created. You can write to these properties with the *Set* function. The following table describes these properties.

Property	Description
<UCPTname>	The name of the Alarm Notifier in the following format: <*network*/*channel*/*device*/*functional block*>.
<UCPTannotation>	The program ID and functional profile template used by the Alarm Notifier. This property is always 8000010128000000[4].UFPTalarmNotifier
<UCPThidden>	A flag indicating whether the Alarm Notifier functional block is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **0** – shown **1** – hidden
<UCPTitemStatus>	This property only appears if the data logger is not synchronized with an LNS network database or it has been deleted. In this case, it has the following values: **IS_NOTSYNCED** **IS_DELETED**
<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.sss]+/-[HH:MM]

Property	Description
<UCPTuri>	The name of the file containing the configuration web page for the Alarm Notifier on the SmartServer flash disk, absolute or relative to /web/user/echelon folder. This property is #8000010128000000[4].UFPTalarmNotifier_Cfg.htm by default.
<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 SmartServer can not exceed the size of the flash memory stored in the SmartServer. The SmartServer will stop writing to the log files when it only has 256 Kb of flash memory remaining.
<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 SmartServer can not exceed the size of the flash memory stored in the SmartServer. The SmartServer 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 SmartServer 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.

Property	Description
<DataPoint>	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 <DataPoint> elements that have an xsi type attribute of "UFPTalarmNotifier_Input_DpRef " and a dpType attribute of "Input". For a description of the properties that must be defined within each <DataPoint> element, see *Input Data Points*. 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*.
<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 Conditions*.
<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 Conditions*.

Property	Description
<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*. Each <AlarmDest> element also contains 2 global elements: its index number (UCPTindex), and its enable data point (UCPTenablePath). The <UCPTenablePath> property is optional. You can reference a **SNVT_switch** data point by its name (UCPTname) 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 Scheduler application. This allows you to enable or disable an Alarm Notifier's destination sets under different circumstances.

7.3.2.1 Input Data Points

The following table describes the properties that you must define within each <DataPoint> element. As described in the previous section, each <DataPoint> element defines an input data point for the Alarm Notifier. The input data points have an xsi type attribute of "UFPTalarmNotifier_Input_DpRef" and a dpType attribute of "Input".

Each time an input data point is 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 logged, and the applicable passive or active alarm destinations will be used.

```
<DataPoint xsi:type="UFPTalarmNotifier_Input_DpRef" dpType="Input" discrim="dir_in" >
    <UCPTname>Net/LON/iLON App/Data Logger[0]/nvoDlLevAlarm[0]</UCPTname>
    <AlarmFlags>
        <UCPTlogEnable>1</UCPTlogEnable>
        <UCPTinvisible>0</UCPTinvisible>
        <UCPTclearRequired>0</UCPTclearRequired>
        <UCPTackRequired>1</UCPTackRequired>
        <UCPTdisabled>0</UCPTdisabled>
        <UCPTcovEnabled>1</UCPTcovEnabled>
    </AlarmFlags>
        <UCPTalarmGroup>0</UCPTalarmGroup>
        <UCPTalarmPriority2>0</UCPTalarmPriority2>
        <UCPTdescription/>
</DataPoint>
```

Property	Description
<UCPTname>	The name of the data point to be monitored by the Alarm Notifier in the following format: <*network*/*channel*/*device*/*functional block*/*data point*>.
<AlarmFlags>	This element contains seven properties that determine what information will be stored in the Alarm History and Alarm Summary Logs for this data point. The meanings of each

Property	Description
	sub-property in the element are described below: **<UCPTlogEnable>:** When this property 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 property 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. **<UCPTinvisible>:** When this property is set to 0, alarm notifications for this data point will be recorded in the Alarm Summary Log. When this property is set to 1, log entries for the data point will not be recorded in the Alarm Summary Log. **<UCPTclearRequired>:** When this property 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 *Write* function. When this property is set to 1, you will need to clear all log entries from the Alarm Summary Log manually with the *Write* function. Note that the <UCPTcovEnabled> property must be set to 0 to record log entries for the data point into the summary log. **<UCPTackRequired>:** When this property is set to 1, all log entries made by the Alarm Notifier for this data point must be manually acknowledged with the *Write* function. When this property 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 the <UCPTclearRequired> property is set to 0. Note that the <UCPTcovEnabled> property must be set to 0 to record log entries for the data point into the summary log. **<UCPTdisabled>:** Set this property to 1 to disable the recording of log entries for the data point. **<UCPTcovEnabled>**: When this property 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 property 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. The default value for all of these properties is 0.
`<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.

Property	Description
<UCPTalarmPriority2>	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 SmartServer Web pages. The default value is 0.
<UCPTdescription>	A user-defined description of the alarm condition for this data point. This can be a maximum of 201 characters long.

7.3.2.2 E-mail Profiles

The following table 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.

```
<Mail>
    <UCPTindex>0</UCPTindex>
    <UCPTnickName>Alarm Notification </UCPTnickName>
    <UCPTemailAddress>user@echelon.com</UCPTemailAddress>
    <UCPTemailFormat>{status}&#13;{new_line}{alarm_time}</UCPTemailFormat>
    <UCPTemailSubject>Data Logger at Alarm Level</UCPTemailSubject>
    <UCPTemailAttachment/>
</Mail>
```

You will reference these e-mail profiles when creating the active and passive destinations 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*.

Property	Description
<UCPTindex>	The index number of the e-mail profile.
<UCPTNickName>	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. The name 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 variable substitutions 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 variable substitutions you can use, see the next section, *E-mail Variable Substitutions.* 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.

Property	Description
<UCPTemailAttachment>	The path of the attachment file that will be sent with the e-mails this profile sends. This must be a file path on the SmartServer flash disk. For example: /root/Data/log1.csv. The path can be a maximum of 1024 characters long.

7.3.2.2.1 E-mail Variable Substitutions

The following table lists the variable substitutions you can use to fill in the <UCPTemailFormat> property within each mail element.

Variable Substitution	Description
{status}	Alarm type name. The enumerated value of the current status of the data point <UCPTpointStatus> that caused the alarm.
{status_code}	Alarm type number. The integer value that maps to the enumerated value of the alarm type stored in <UCPTpointStatus>. For example, the integer value of AL_ALM_CONDITION is 5.
{alarm_date}	Alarm Date. The date the alarm occurred, expressed in the following format: YYYY-MM-DD. For example: 2002-30-10
{alarm_day}	Alarm Date Day. The day the alarm occurred, as an integer between 1 and 31.
{alarm_month}	Alarm Date Month. The month the alarm occurred, as an integer between 1 and 12.
{alarm_year}	Alarm Date Year. The year the alarm was triggered.
{alarm_group}	Alarm Group Number. This is determined by the <UCPTalarmGroup> property assigned to the data point that caused the alarm within the Alarm Notifier.
{alarm_limit}	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.
{dp_alias_name}	Alarm Location String. The <UCPTaliasName> property of the data point that generated the alarm.
{nv_index}	Object Number or NV Index. The index of the network variable associated with the data point that triggered the alarm.
{alarm_priority}	Alarm priority. The priority of the alarm, as specified by <UCPTalarmPriority2>.
{snvt_ID}	Alarm SNVT ID. The index defined for the standard network variable type (SNVT) used by the data point, as defined in the LonMark Resource files.

Variable Substitution	Description
{alarm_hour_12}	Alarm Time Hour (12 hour). The time the alarm was triggered on a 12-hour clock. For example, this would return 10 for an alarm that occurred at 10:00 AM or 10:00 PM.
{alarm_hour}	Alarm Time Hour (24 hour). The time the alarm was triggered on a 24-hour clock. For example, this would return 16 for an alarm that occurred at 4 PM.
{alarm_time_am_pm}	Alarm time AM/PM. Returns “AM” for alarms that occurred in the morning, or “PM” for alarms that occurred in the afternoon.
{alarm_time}	Alarm Time. 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.
{alarm_minute}	Alarm Time Minutes. The minute when the alarm was triggered.
{alarm_second}	Alarm Time Seconds. The second when the alarm was triggered.
{alarm_millisecond}	Alarm Time Milliseconds. The millisecond when the alarm was triggered.
{dp_value}	Alarm Value. The <UCPTvalue> of the data point that triggered the alarm expressed with the LonFormat="#<*programID*> [*scope*].<*data type*>" attribute. This information is typically received on the **SNVT_alarm** input.
{dp_preset}	Data Point Preset. The <UCPTvalue> property of the data point expressed with the LonFormat="UCPTvalueDef" attribute. This is the value defined for the data point by a preset.
{alarm_description}	Alarm Description. The description of the alarm that triggered the alarm, as specified in <UCPTdescription>.
{dp_name}	Input Point Name. The name of the data point defined in <UCPTname>.
{dp_description}	Input Point Description. The description of the data point that generated the alarm as specified in <UCPTdescription>.
{dp_unit}	Alarm Value Unit. The units of the data point that generated the alarm as defined in <UCPTunit>. If the data point is a **SNVT_alarm** or **SNVT_alarm_2** type (you are using the Alarm Notifier in conjunction with the Alarm Generator), SI units will be returned by this variable, regardless of the data point's format.

Variable Substitution	Description
{dp_value_alias}	Data Point Value (by Alias Name). The <UCPTaliasName> of the data point and its <UCPTvalue> at the time that the alarm was triggered. The <UCPTvalue> property is expressed with the LonFormat="#<*programID*> [*scope*].<*data type*>" attribute. This is the format defined by the data type used by the data point.
{dp_preset_alias}	Data Point Preset (by Alias Name). The <UCPTaliasName> of the data point and its <UCPTvalue> at the time that the alarm was triggered. The <UCPTvalue> property is expressed with the LonFormat="UCPTvalueDef" attribute. This is the value defined for the data point by a preset.
{ip_address}	IP Address of the Network Interface. The IP address of the network interface used to send the e-mail.
{new_line}	Line feed. Inserts a carriage return into your e-mail.

7.3.2.3 Active and Passive Alarm Conditions

The following table 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 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 a passive alarm, and the passive alarm destinations will be used for the alarm notification.

```
<ActiveAlarm>
    <UCPTindex>5</UCPTindex>
    <UCPTlevel>240</UCPTlevel>
    <UCPTalarmText>Alarm</UCPTalarmText>
    <UCPTalarmCondition LonFormat="UCPTalarmCondition">AL_ALM_CONDITION</UCPTalarmCondition>
</ActiveAlarm>
```

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

Property	Description
`<UCPTindex>`	The index number of the alarm condition.
`<UCPTlevel>`	Enter an alarm level for the condition set, in the range 0-255. The level assigned to a condition 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 for the alarm conditions using levels 0-125, and another for the alarm conditions using levels 126-255. Alarm conditions assigned levels 0-125 would use the first destination, and alarm conditions

Property	Description
	assigned level 126-255 would use the second destination. **NOTE:** If you use the Configuration Plug-In 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.
`<UCPTalarmCondition>`	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 condition type. The valid alarm type identifiers are as follows: 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 conditions, and more severe conditions such as AL_HIGH_LMT_ALM_1 for your active conditions.

7.3.2.4 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.

```
<AlarmDest>
    <UCPTindex>0</UCPTindex>
    <UCPTenablePath/>
    <ActiveDest>
            <UCPTmailPath>Mail[UCPTnickName="Alarm Notification"]</UCPTmailPath>
```

```
            <UCPTdataPointPath>DataPoint[@dpType="Output" and UCPTnickName="Net/LON/iLON App/Data
            Logger[0]/nviDlClear[0]"]</UCPTdataPointPath>
            <UCPTpointValue>ON</UCPTpointValue>
            <UCPTminLevel>255</UCPTminLevel>
            <UCPTmaxLevel>0</UCPTmaxLevel>
            <UCPTnackDelay>0</UCPTnackDelay>
    </ActiveDest>
    <PassiveDest>
            <UCPTminLevel>255</UCPTminLevel>
            <UCPTmaxLevel>0</UCPTmaxLevel>
            <UCPTnackDelay>0</UCPTnackDelay>
    </PassiveDest>
</AlarmDest>
```

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 be 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 Scheduler application. In this fashion, you can enable or disable destination sets as you like.

Every <AlarmDest> should contain zero or one <ActiveDest> and zero or one <PassiveDest> elements. The following table describes the properties you must define for each <ActiveDest> and <PassiveDest> element. Each <ActiveDest> element defines an active destination for the Alarm Notifier. Each <PassiveDest> element defines a passive destination 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 conditions. 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 conditions.

Property	Description
<UCPTmailPath>	Contains an e-mail nickname, as defined for an e-mail profile created for the Alarm Notifier. This signifies the e-mail profile to be used each time an alarm notification uses this destination. The string must be a valid *xPath* expression that points to an e-mail profile. The e-mail will be sent after <UCPTnackDelay> expired.
<UCPTdataPointPath>	Contains the xPath to the <DataPoint> with dpType="Output" 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 <UCPTnackDelay> property. This is only applicable if the alarm has not been acknowledged.
<UCPTpointValue>	The value or preset to be written to the output data point specified by <UCPTdataPointPath>.
<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 the alarm.

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.

7.3.3 Using the Set Function on an Alarm Notifier

Use the *Set* function to create new Alarm Notifiers, or to overwrite the configuration of existing Alarm Notifiers. The Alarm Notifiers to be created or written to are signified by a list of <Item> elements in the input parameters supplied to the function. The properties you must define within each <Item> element are the same, whether you are creating a new Alarm Notifier or modifying an existing Alarm Notifier. The previous section, *Using the Get Function on an Alarm Notifier*, describes these properties.

Note: If you specify an Alarm Notifier with the <UCPTname> element, the *Set* function deletes the specified Alarm Notifier before the specified parameters are set. If the <UCPTname> element is not specified, a new Alarm Notifier is created.

When modifying an existing Alarm Notifier, any optional properties omitted from the *Set* Request, such as the input point points, will be erased. Old values will not be preserved, so you must fill in every property when writing to an Alarm Notifier, even if you are not changing all of the values.

The first invocation of the *Set* function will generate the #8000010128000000[4].UFPTalarmNotifier.xml file in the root/config/network/<network>/<channel>/iLONApp ||<device> directory of the SmartServer, if the file does not already exist.

When creating or modifying an Alarm Notifier with the *Set* function, you may want to use output from the *Get* function as the basis for your input. 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 example below creates an Alarm Notifier that uses an **nviRequest** 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 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 and the alarm level assigned to the alarm condition set.

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTalarmNotifier_Cfg">
        <UCPTname>Net/LON/iLON App/Alarm Notifier[0]</UCPTname>
        <DataPoint dpType="nviEnable" discrim="dir_in">
            <UCPTname>Net/LON/iLON App/Alarm Notifier[0]/nviAnEnable[0]</UCPTname>
        </DataPoint>
        <DataPoint xsi:type="UFPTalarmNotifier_Input_DpRef" dpType="Input" discrim="dir_in">
            <UCPTname>Net/LON/iLON App/VirtFb/temp_f</UCPTname>
        <AlarmFlags>
            <UCPTlogEnable>1</UCPTlogEnable>
            <UCPTinvisible>0</UCPTinvisible>
            <UCPTclearRequired>0</UCPTclearRequired>
            <UCPTackRequired>1</UCPTackRequired>
            <UCPTdisabled>0</UCPTdisabled>
            <UCPTcovEnabled>1</UCPTcovEnabled>
        </AlarmFlags>
        <UCPTalarmGroup>0</UCPTalarmGroup>
        <UCPTalarmPriority2>0</UCPTalarmPriority2>
```

```
            <UCPTdescription />
        </DataPoint>
        <DataPoint xsi:type="UFPTalarmNotifier_DpRef" dpType="Output">
            <UCPTname>Net/LON/iLON App/VirtFb/temp_f</UCPTname>
            <UCPTnickName>Net/LON/iLON App/VirtFb/temp_f</UCPTnickName>
        </DataPoint>
        <DataPoint xsi:type="UFPTalarmNotifier_DpRef" dpType="Output">
            <UCPTname>Net/LON/iLON App/Digital Output 1/nviClaValue_1</UCPTname>
            <UCPTnickName>Net/LON/iLON App/Digital Output 1/nviClaValue_1</UCPTnickName>
        </DataPoint>
        <DataPoint xsi:type="UFPTalarmNotifier_DpRef" dpType="Output">
            <UCPTname>Net/LON/iLON App/Digital Output 2/nviClaValue_2</UCPTname>
            <UCPTnickName>Net/LON/iLON App/Digital Output 2/nviClaValue_2</UCPTnickName>
        </DataPoint>
        <SCPTdelayTime>0</SCPTdelayTime>
        <UCPTsumLogSize>50</UCPTsumLogSize>
        <UCPThistLogSize>100</UCPThistLogSize>
        <UCPTlogFormat xsi:type="string" LonFormat="UCPTlogFormat">LF_TEXT</UCPTlogFormat>
        <UCPTsumLogFileName>Net/LON/iLON App/Alarm Notifier[0]_Summary.csv</UCPTsumLogFileName>
        <UCPThistLogFileName>Net/LON/iLON App/Alarm Notifier[0]_History.csv</UCPThistLogFileName>
        <UCPTemailAggregTime>0</UCPTemailAggregTime>
        <ActiveAlarm>
            <UCPTindex>5</UCPTindex>
            <UCPTlevel>240</UCPTlevel>
            <UCPTalarmText>Alarm</UCPTalarmText>
            <UCPTalarmCondition LonFormat="UCPTalarmCondition">AL_ALM_CONDITION</UCPTalarmCondition>
        </ActiveAlarm>
        <PassiveAlarm>
            <UCPTindex>0</UCPTindex>
            <UCPTlevel>255</UCPTlevel>
            <UCPTalarmText>Online</UCPTalarmText>
            <UCPTalarmCondition LonFormat="UCPTalarmCondition">AL_NO_CONDITION</UCPTalarmCondition>
        </PassiveAlarm>
        <AlarmDest>
            <UCPTindex>0</UCPTindex>
            <UCPTenablePath xsi:type="string" />
            <ActiveDest xsi:type="UFPTalarmNotifier_CfgAlarmDestination">
                <UCPTdataPointPath xsi:type="string">DataPoint[@dpType="Output" and
                   UCPTnickName="Net/LON/iLON App/Digital Output 1/nviClaValue_1"]
                </UCPTdataPointPath>
                <UCPTpointValue>ON</UCPTpointValue>
                <UCPTminLevel>255</UCPTminLevel>
                <UCPTmaxLevel>0</UCPTmaxLevel>
                <UCPTnackDelay>0</UCPTnackDelay>
            </ActiveDest>
            <PassiveDest xsi:type="UFPTalarmNotifier_CfgAlarmDestination">
                <UCPTdataPointPath xsi:type="string">
                    DataPoint[@dpType="Output" and UCPTnickName="Net/LON/iLON App/Digital Output
                    2/nviClaValue_2"]
                </UCPTdataPointPath>
                <UCPTpointValue>OFF</UCPTpointValue>
                <UCPTminLevel>255</UCPTminLevel>
                <UCPTmaxLevel>0</UCPTmaxLevel>
                <UCPTnackDelay>0</UCPTnackDelay>
            </PassiveDest>
        </AlarmDest>
    </Item>
  </iLonItem>
</Set>
```

7.3.4 Using the Read Function on an Alarm Notifier

Each time an Alarm Notifier causes an alarm notification, it will record an entry for the notification into its log file. You can use the *Read* function to retrieve the log entries that an Alarm Notifier has recorded. You must reference the Alarm Notifier to return log entries for by its <UCPTname>in the input you supply to the function.

The alarm log files are stored in the `/root/AlarmLog` directory of the SmartServer. These files are named histlog*X*, 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. You should not attempt to read more than 100 log entries with a single *Read* request.

The following code demonstrates how to use a *Read* request to generate a list of up to the first 100 entries stored in a specific Alarm Log.

Request

```
<Read xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="UFPTalarmNotifier_Data"][UCPTalarmLog="HISTORICAL"]
                  [position()&lt;100]
    </xSelect>
    <Item>
        <UCPTname>Net/LON/iLON App/Alarm Notifier[0]</UCPTname>
    </Item>
  </iLonItem>
</Read>
```

Response

```
<ReadResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="UFPTalarmNotifier_Meta_Data" >
        <UCPTname>Net/LON/iLON App/Alarm Notifier[0]</UCPTname>
        <UCPTlastUpdate>2008-02-29T15:01:40.650-08:00</UCPTlastUpdate>
        <UCPTalarmLog LonFormat="UCPTalarmLog">HISTORICAL</UCPTalarmLog>
        <UCPTstart>2008-02-29T11:14:41.300-08:00</UCPTstart>
        <UCPTstop>2008-02-29T11:14:41.300-08:00</UCPTstop>
        <UCPTmodificationNumber>0</UCPTmodificationNumber>
        <UCPTlogLevel>0.496</UCPTlogLevel>
        <UCPTtotalCount>1</UCPTtotalCount>
    </Item>
    <Item xsi:type="UFPTalarmNotifier_Data">
        <UCPTname>Net/LON/iLON App/Data Logger[0]/nvoDlLevAlarm[0]</UCPTname>
        <UCPTaliasName>nvoDlLevAlarm[0]</UCPTaliasName>
        <UCPTlastUpdate>2008-02-29T11:14:41.300-08:00</UCPTlastUpdate>
        <UCPTdescription> </UCPTdescription>
        <UCPTvalue LonFormat="#0000000000000000[0].SNVT_alarm#LO">0</UCPTvalue>
        <UCPTpointStatus LonFormat="UCPTpointStatus">AL_NO_CONDITION</UCPTpointStatus>
        <UCPTpriority>255</UCPTpriority>
        <UCPTmetaDataPath>//*[@xsi:type="UFPTalarmNotifier_Meta_Data"]
                            [UCPTname="Net/LON/iLON App/Alarm Notifier[0]"]
        </UCPTmetaDataPath>
        <UCPTalarmNotifierName>Net/LON/iLON App/Alarm Notifier[0]</UCPTalarmNotifierName>
        <UCPTalarmTime>2008-02-29T11:13:44.560-08:00</UCPTalarmTime>
        <UCPTuserName>iLON</UCPTuserName>
        <UCPTstatus LonFormat="UCPTalarmStatus">AUTO_CLEAR</UCPTstatus>
        <ReadData>
            <UCPTalarmText>Online</UCPTalarmText>
            <UCPTalarmPriority2>0</UCPTalarmPriority2>
            <UCPTalarmGroup>0</UCPTalarmGroup>
            <UCPTalarmType LonFormat="UCPTalarmType">PASSIVE</UCPTalarmType>
        </ReadData>
    </Item>
  </iLonItem>
</ReadResponse>
```

In addition to the requested log entries, the *Read* function returns a single <Item> of type "UFPTalarmNotifier_Meta_Data" for each log file from which entries were read. This "UFPTalarmNotifier_Meta_Data" <Item> has the following properties:

`<UCPTname>`	The name of the data logger from which entries were read in the following format: <*network*/*channel*/*device*/*functional block*>.
`<UCPTlastUpdate>`	A timestamp indicating the time that the last log entry was

	made.
<UCPTalarmLog>	The type of log requested (either HISTORICAL or SUMMARY).
<UCPTstart> <UCPTstop>	Timestamps indicating the log times of the first and last log entries in the log file.
<UCPTmodificationNumber>	A counter indicating the number of times the log file has been modified. The counter is not increased when data is added to the end of the log, but only if some modifications are made to the existing data.
<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 *Read* function returns an <Item> element of type "UFPTalarmNotifier _Data" for each log entry that met the selection criteria you defined in the input parameters log file from which entries were read. Each "UFPTalarmNotifier_Data" <Item> has the following properties:

Property	Description
<UCPTname>	The name of the data point in the following format: <*network*/*channel*/*device*/*functional block*/*data point*>.
<UCPTaliasName>	The default or user-defined nickname provided for the data point.
<UCPTlastUpdate>	A timestamp indicating the time that the log entry was made.
<UCPTdescription>	The comment entered into the log entry for the log. You can enter comments into the log with the *Write* function.
<UCPTvalue>	The value the data point was updated to when the log entry was made. The value may be presented in the following two LonFormats: • LonFormat="#<*programID*>[*scope*].<*data type*>". The format is specified by the data type defined for the data point. For a **SNVT_switch** data point, this value could be 100.0 1 or 0.0 0, for example. • LonFormat="UCPTvalueDef". The value defined for the data point by a preset. For a **SNVT_switch** data point, this value could be ON or OFF, for example. If a preset is not defined for the data point, this value is AL_NUL.
<UCPTpointStatus>	The status the data point was updated to when the log entry was made.

Property	Description
<UCPTpriority>	The priority level currently assigned to the data point (0-255). The priority level of a data point determines which applications can write to its value. You can modify the value of this property with the *Write* or *ResetPriority* functions.
<UCPTalarmNotifierName>	The name of the alarm notifier in which the log entry is stored in the following format: <*network*/*channel*/*device*/*functional block*>.
<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.
<UCPTuserName>	The name of the user who acknowledged the alarm. Alarms can be acknowledged with the *Write* function.
<UCPTalarmStatus>	The status of the alarm. The status can initially be AUTO_ACK, AUTO_ACK, or AUTO_CLEAR, depending on the flags that were set. The status can be changed to MANUAL_ACK, MANUAL_CLEAR, MANUAL_ACK_MANUAL_CLEAR, MANUAL_ACK_AUTO_CLEAR, AUTO_ACK_AUTO_CLEAR or AUTO_ACK_MANUAL_CLEAR. You can clear or acknowledge alarms manually with the *Clear* function. For more information, see *Using the Clear Function on an Alarm Notifier Log File*. Alarms may be cleared or acknowledged automatically depending on how the <UCPTflags> property was defined for the Alarm Notifier when it was created.
<ReadData>	• This element contains the following properties: • <UCPTalarmText>. The alarm text for the alarm. You can specify this text with the *Set* function. • <UCPTalarmPriority2>. The priority level that the Alarm Notifier application is using to update the value of the data point. The Alarm Notifier will only successfully update the value of the data point if it is using a priority level higher than (or equal to) the priority assigned to the data point in the Data Server. • <UCPTalarmGroup>. The alarm group of the alarm. This may be useful when sorting alarms. • <UCPTalarmType>. ACTIVE or PASSIVE. This indicates whether the alarm was an active or passive alarm. The conditions that determine this are defined when the Alarm Notifier is created. For more information, see *Active and Passive Alarm Conditions*.

7.3.5 Using the Write Function on an Alarm Notifier Log File

You can use the *Write* function to acknowledge or write a comment to a log entry on an Alarm Notifier. The following table describes the properties you can define in the input parameters you supply to the function to acknowledge an alarm.

Request

```
<Write xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTalarmNotifier_Data">
        <UCPTname>Net/LON/iLON App/Digital Input 1/nvoClsValue_1</UCPTname>
        <UCPTdescription>enter new comment</UCPTdescription>
        <UCPTalarmNotifierName>Net/LON/iLON App/Alarm Notifier[0]</UCPTalarmNotifierName>
        <UCPTalarmTime>2008-02-29T11:13:44.560-08:00</UCPTalarmTime>
        <UCPTuserName>iLON</UCPTuserName>
        <UCPTstatus xsi:type="string" LonFormat="UCPTalarmStatus">MANUAL_CLEAR</UCPTstatus>
        </Item>
  </iLonItem>
</Write>
```

Property	Description
<UCPTname>	The name of the data point that triggered the alarm in the following format: <*network*/*channel*/*device*/*functional block*/*data point*>.
<UCPTdescription>	Enter a comment to be recorded in the log file entry for this alarm. This can be a maximum of 80 characters long.
<UCPTalarmNotifier Name>	The name of the alarm notifier containing the data point that triggered the alarm in the following format: <*network*/*channel*/*device*/*functional block*>.
<UCPTalarmTime>	A timestamp indicating the time that the alarm occurred. You must enter this timestamp in ISO 8601 format, which is as follows: [YYYY-MM-DD]T[HH:MM:SS.sss]+/-[HH:MM]
<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.
<UCPTStatus LonFormat= "UCPTalarmStatus">	You can select one of four parameters for the <UCPTalarmStatus> attribute: • **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. If an alarm is cleared manually before it is acknowledged manually, the log entry is set to MANUAL_ACK and deleted from the alarm summary log (cleared = MANUAL_CLEAR). • **NACK:** The 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:** 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 <UCPTackRequired> property for the data point to 1 when you create your Alarm Notifier with *Set*. You can still enter comments for the log file using this function if an alarm was automatically acknowledged. For more information on the

Property	Description
	<UCPTackRequired> property, see *Input Data Points*. • **AUTO_CLEAR:** The alarm was automatically cleared from the alarm log when it occurred. You can cause an Alarm Notifier to automatically clear all alarms for a data point by setting <UCPTclearRequired> property for the data point to 1 when you create your Alarm Notifier with *Set*. • **MANUAL_ACK_MANUAL_CLEAR:** The Alarm has not been acknowledged or removed from the active list. • **MANUAL_ACK_AUTO_CLEAR:** The Alarm was not acknowledged, but it was automatically cleared from the alarm log when it occurred. • **AUTO_ACK_AUTO_CLEAR:** The alarm was automatically acknowledged and cleared by the Alarm Notifier when it occurred. • **AUTO_ACK_MANUAL_CLEAR:** The alarm was automatically acknowledged by the Alarm Notifier, but it was not cleared.

Response

```
<WriteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="UFPTalarmNotifier_Data" >
        <UCPTname>Net/LON/iLON App/Digital Input 1/nvoClsValue_1</UCPTname>
        <UCPTalarmNotifierName>Net/LON/iLON App/Alarm Notifier[0]</UCPTalarmNotifierName>
        <UCPTalarmTime>2008-02-29T17:47:53.601-08:00</UCPTalarmTime>
    </Item>
  </iLonItem>
</WriteResponse>
```

7.3.6 Using the Clear Function on an Alarm Notifier Log File

You can use the *Clear* function to remove log entries from an Alarm Notifier's log file. You can specify which Alarm Notifier is to be affected, and which log entries will be removed using xSelect statements. If no filter is specified with an xSelect statement, the whole alarm log will be deleted.

Note: This function only deletes the log entries. You can delete the Alarm Notifier itself using the *Delete* function.

The following call to the *Clear* function deletes up to 100 log entries from an alarm notifier log that occurred before the specified date and time.

Request

```
<Clear xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
    <xSelect> //Item[UCPTlastUpdate&gt;="2008-03-03T10:38:30.000-8:00"]
                    [UCPTlastUpdate&lt;="2008-03-03T11:38:29.240-8:00"]
                    [position()&lt;=99]
                    [UCPTalarmLog="HISTORICAL"]
    </xSelect>
    <Item>
        <UCPTname>Net/LON/iLON App/Alarm Notifier[0]</UCPTname>
    </Item>
  </iLonItem>
</Clear>
```

Response

```
<ClearResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="UFPTalarmNotifier_ClearResponse" >
        <UCPTname>Net/LON/iLON App/Alarm Notifier[0]</UCPTname>
        <UCPTlastUpdate>2008-03-03T11:39:21.960-08:00</UCPTlastUpdate>
        <UCPTalarmLog LonFormat="UCPTalarmLog">HISTORICAL</UCPTalarmLog>
        <UCPTstart>2008-02-29T17:50:00.580-08:00</UCPTstart>
        <UCPTstop>2008-02-29T17:50:00.580-08:00</UCPTstop>
        <UCPTmodificationNumber>4</UCPTmodificationNumber>
        <UCPTlogLevel>0.495</UCPTlogLevel>
        <UCPTtotalCount>1</UCPTtotalCount>
    </Item>
  </iLonItem>
</ClearResponse>
```

The *Clear* function returns the following information related to the alarm log file from which entries were deleted:

Property	Description
<UCPTname>	The name of the alarm log in which entries were cleared in the following format: <*network*/*channel*/*device*/*functional block*>.
<UCPTlastUpdate>	A timestamp indicating the time that the log was cleared.
<UCPTalarmLog>	The type of log file affected by the function: SUMMARY or HISTORICAL.
<UCPTfileName>	The name of the log file the Alarm log is using.
<UCPTstart> <UCPTstop>	Timestamps indicating the times of the first and last log entries in the log file.
<UCPTmodificationNumber>	A counter indicating the number of times the log file has been modified. The counter is not increased when data is added to the end of the log, but only if some modifications are made to the existing data.
<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	The total number of entries contained in the data log read by the function.

7.3.7 Using the Delete Function on an Alarm Notifier

You can use the *Delete* function to delete an Alarm Notifier. To delete an Alarm Notifier, you provide an <Item> element with a UFPTalarmNotifier _Cfg type that includes the <UCPTname> property of the alarm notifier to be deleted. The following code sample demonstrates how to use the *Delete* function to delete an Alarm Notifier:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <Item xsi:type="UFPTalarmNotifier_Cfg">
            <UCPTname> Net/LON/iLON App/Alarm Notifier[0]</UCPTname>
        </Item>
```

```
    </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/LON/iLON App/Alarm Notifier[0]</UCPTname>
    </Item>
 </iLonItem>
</DeleteResponse>
```

8 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 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.

8.1 Overview of the AnalogFB XML File

The #8000010128000000[4].UFPTanalogFunctionBlock.xml file stores the configuration of the Analog Function Blocks that you have added to the SmartServer.

You can create new Analog Functional Blocks using the *Set* function, or by manually editing the #8000010128000000[4].UFPTanalogFunctionBlock.xml, and rebooting the SmartServer. You can create up to 20 Analog Functional Blocks per SmartServer. You can add more than 20 Analog Functional Blocks if you load the dynamic v40 XIF on your SmartServer and you operate your SmartServer in Standalone mode. Note that using the v40 XIF with the SmartServer operating in LNS mode (**LNS Auto** or **LNS Manual**) is not supported.

The following represents a sample #8000010128000000[4].UFPT #8000010128000000[4].UFPTanalogFunctionBlock.xml file with one Analog Functional Block. This Analog Function Block determines the maximum value of the value field of the **nviClaValue_1** and **nviClaValue_2** data points on the **Digital Output 1** and **Digital Output 2** functional blocks on the SmartServer, and stores that value in the value field of the **nvoClsValue_1** data point on the **Digital Input 1** functional block on the SmartServer.

```
<Item xsi:type="UFPTanalogFunctionBlock_Cfg">
    <UCPTname>Net/LON/iLON App/Analog Fn Block[0]</UCPTname>
    <UCPTannotation>8000010128000000[4].UFPTanalogFunctionBlock</UCPTannotation>
    <UCPThidden>0</UCPThidden>
    <UCPTlastUpdate>2008-03-03T12:25:33.900-08:00</UCPTlastUpdate>
    <UCPTuri>#8000010128000000[4].UFPTanalogFunctionBlock_Cfg.htm</UCPTuri>
    <DataPoint dpType="nvoDropOut" discrim="dir_out" >
        <UCPTname>Net/LON/iLON App/Analog Fn Block[0]/nvoAfbDropOut[0]</UCPTname>
    </DataPoint>
    <DataPoint xsi:type="UFPTanalogFunctionBlock_DpRef" dpType="Input" discrim="dir_in" >
        <UCPTname>Net/LON/iLON App/Digital Output 1/nviClaValue_1</UCPTname>
        <UCPTfieldName>value</UCPTfieldName>
        <UCPTpollRate>900</UCPTpollRate>
```

```
    </DataPoint>
    <DataPoint xsi:type="UFPTanalogFunctionBlock_DpRef" dpType="Input" discrim="dir_in" >
        <UCPTname>Net/LON/iLON App/Digital Output 2/nviClaValue_2</UCPTname>
        <UCPTfieldName>value</UCPTfieldName>
        <UCPTpollRate>900</UCPTpollRate>
    </DataPoint>
    <DataPoint xsi:type="UFPTanalogFunctionBlock_DpRef" dpType="Output" discrim="dir_out" >
        <UCPTname>Net/LON/iLON App/Digital Input 1/nvoClsValue_1</UCPTname>
        <UCPTfieldName>value</UCPTfieldName>
    </DataPoint>
    <UCPTcompFunction LonFormat="UCPTcompFunction" >FN_NUL</UCPTcompFunction>
    <UCPTmajorityValue>100</UCPTmajorityValue>
    <UCPTtrueThreshold LonFormat="UNVT_float" ></UCPTtrueThreshold>
    <UCPToutputFunction LonFormat="UCPToutputFunction" >FN_MAX</UCPToutputFunction>
    <SCPTminRnge LonFormat="#0000000000000000[0].SNVT_switch.value" >0</SCPTminRnge>
    <SCPTmaxRnge LonFormat="#0000000000000000[0].SNVT_switch.value" >0</SCPTmaxRnge>
    <UCPTcalculationInterval>0.0</UCPTcalculationInterval>
    <SCPTovrBehave LonFormat="SCPTovrBehave" >OV_RETAIN</SCPTovrBehave>
    <UCPTpollOnResetDelay>0.0</UCPTpollOnResetDelay>
</Item>
```

8.2 Creating and Modifying the Analog Functional Block XML File

You can create and manage the #8000010128000000[4].UFPTanalogFunctionBlock.xml file with the *Set* SOAP function. The following section, *Analog Functional Block SOAP Interface*, describes how to use *Set* and the other SOAP functions provided for the Analog Functional Block application.

Alternatively, you can create and manage the #8000010128000000[4].UFPTanalogFunctionBlock.xml file manually with an XML editor and download it to the SmartServer via FTP. Echelon does not recommend this, as the SmartServer will require a reboot to read the configuration of the downloaded file. Additionally, the SmartServer 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 Analog Functional Block's configuration.

8.3 Analog Functional Block SOAP Interface

You can use the SOAP interface to perform the following functions on an Analog Functional Block application:

Function	Description
List	Generate a list of the Analog Functional Blocks that you have added to the SmartServer.
Get	Retrieve the configuration of any Analog Functional Block that you have added to the SmartServer.
Set	Create a new Analog Functional Block, or overwrite the configuration of an existing Analog Functional Block.
Delete	Delete an Analog Functional Block.

8.3.1 Using the List Function on an Analog Functional Block

Use the *List* function to retrieve a list of the Analog Functional Blocks that you have added to the SmartServer. The List function takes an <iLonItem> element that includes an xSelect statement

querying items of a UFPTanalogFunctionBlock_Cfg type as its input, as shown in the example below. The *List* function returns an <Item> element for each Analog Functional Block that you have added to the SmartServer. The next section describes the properties included in each of these elements.

You could use the list of <Item> elements returned by this function as input for the *Get* function. The *Get* function would then return the configuration of each Analog Functional Block included in the list.

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <xSelect>//Item[@xsi:type="UFPTanalogFunctionBlock_Cfg"]</xSelect>
    </iLonItem>
</List>
```

Response

```
<ListResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>Net/LON/iLON App/Analog Fn Block[0]</UCPTname>
        <UCPTannotation>#8000010128000000[4].UFPTanalogFunctionBlock;xsi:type="LON_Fb_Cfg"
        </UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
  </iLonItem>
</ListResponse>
```

8.3.2 Using the Get Function on an Analog Functional Block

You can use the *Get* function to retrieve the configuration of any Analog Functional Block that you have added to the SmartServer. You must reference the Analog Functional Block whose configuration is to be returned by its <UCPTname> in the input you supply to the function, as shown in the example below.

Request

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>Net/LON/iLON App/Analog Fn Block[0]</UCPTname>
    </Item>
  </iLonItem>
</Get>
```

Response

```
GetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="UFPTanalogFunctionBlock_Cfg">
        <UCPTname>Net/LON/iLON App/Analog Fn Block[0]</UCPTname>
        <UCPTannotation>8000010128000000[4].UFPTanalogFunctionBlock</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-08-06T16:43:06.160-07:00</UCPTlastUpdate>
        <UCPTuri>#8000010128000000[4].UFPTanalogFunctionBlock_Cfg.htm</UCPTuri>
        <DataPoint dpType="nvoDropOut" discrim="dir_out">
            <UCPTname>Net/LON/iLON App/Analog Fn Block[0]/nvoAfbDropOut[0]</UCPTname>
        </DataPoint>
        <DataPoint xsi:type="UFPTanalogFunctionBlock_DpRef" dpType="Input" discrim="dir_in">
            <UCPTname>Net/LON/iLON App/Digital Output 1/nviClaValue_1</UCPTname>
            <UCPTfieldName>value</UCPTfieldName>
            <UCPTpollRate>900</UCPTpollRate>
        </DataPoint>
        <DataPoint xsi:type="UFPTanalogFunctionBlock_DpRef" dpType="Input" discrim="dir_in">
            <UCPTname>Net/LON/iLON App/Digital Output 2/nviClaValue_2</UCPTname>
            <UCPTfieldName>value</UCPTfieldName>
            <UCPTpollRate>900</UCPTpollRate>
        </DataPoint>
        <DataPoint xsi:type="UFPTanalogFunctionBlock_DpRef" dpType="Output" discrim="dir_out">
```

```
            <UCPTname>Net/LON/iLON App/Digital Input 1/nvoClsValue_1</UCPTname>
            <UCPTfieldName>value</UCPTfieldName>
         </DataPoint>
         <UCPTcompFunction LonFormat="UCPTcompFunction">FN_NUL</UCPTcompFunction>
         <UCPTmajorityValue>100</UCPTmajorityValue>
         <UCPTtrueThreshold LonFormat="UNVT_float" />
         <UCPToutputFunction LonFormat="UCPToutputFunction">FN_AVERAGE</UCPToutputFunction>
         <SCPTminRnge LonFormat="#0000000000000000[0].SNVT_switch.value">0</SCPTminRnge>
         <SCPTmaxRnge LonFormat="#0000000000000000[0].SNVT_switch.value">100</SCPTmaxRnge>
         <UCPTcalculationInterval>0.0</UCPTcalculationInterval>
         <UCPTcalculationMatrix>1</UCPTcalculationMatrix>
         <SCPTovrBehave LonFormat="SCPTovrBehave">OV_RETAIN</SCPTovrBehave>
         <UCPTpollOnResetDelay>0.0</UCPTpollOnResetDelay>
      </Item>
   </iLonItem>
</GetResponse></SOAP-ENV:Body></SOAP-ENV:Envelope>
```

The function returns an <Item> element for each Analog Functional Block referenced in the input parameters supplied to the function. The properties included in each element are initially defined when the Analog Functional Block is created. You can write to these properties with the *Set* function. The following table describes these properties.

Property	Description
<UCPTname>	The name of the Analog Functional Block in the following format: <*network*/*channel*/*device*/*functional block*>.
<UCPTannotation>	The program ID and functional profile template used by the Analog Functional Block. This property is always 8000010128000000[4].UFPTanalogFunctionBlock.
<UCPThidden>	A flag indicating whether the Analog Functional Block functional block is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **0** – shown **1** – hidden
<UCPTitemStatus>	This property only appears if the data logger is not synchronized with an LNS network database or it has been deleted. In this case, it has the following values: **IS_NOTSYNCED** **IS_DELETED**
<UCPTlastUpdate>	A timestamp indicating the last time the configuration of the Analog Functional Block was updated. This timestamp uses the following format: YYYY-MM-DD**T**HH:MM:SS**Z**
<UCPTuri>	The name of the file containing the configuration web page for the Analog Functional Block on the SmartServer flash disk, absolute or relative to /web/user/echelon. This property is #8000010128000000[4].UFPTanalogFunctionBlock_Cfg by default.
<DataPoint>	You can specify as many input data points as you want per

Property	Description
`Input`	Analog Function Block. The input data points for an Analog Function Block are defined by a list of <DataPoint> elements with dpType attributes of ="Input". 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 SmartServer 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 an Alarm Generator 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.
`<DataPoint>` `Compare`	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

Property	Description
	<UCPTtrueThreshold> property is defined.
<DataPoint> Output	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.
<UCPTcompFunction>	This property defines the comparison function the Analog Function Block will use to compare the values of the compare and input data points. This function will only be used if the <UCPToutputFunction> property is set to 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*.
<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 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. This field has a range of 0.0 to 100.0. 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.
<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 be made with the value of the compare data point. You can

Property	Description
	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 are included in the *Comparison Functions* section.
<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*.
<SCPTminRnge>	The minimum value that the output data point can be assigned.
<SCPTmaxRnge>	The maximum value that the output data point can be assigned.
<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 Analog Function Block will update the output data point each time any of the input data points are updated.
<UCPTcalculationMatrix>	The number of previous averages from which a straight average is calculated. For example, if you set this property to 2, the average equals the sum of the last two calculated averages divided by two. This means that if the last calculated average was 20, and the previous calculated average was 30, the average value sent to the output point is 25. **Note**: This property is only applicable if you set <UCPToutputFunction> to FN_AVERAGE.
<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 output data point will maintain its last setting when an override occurs.

Property	Description
<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.

8.3.2.1 Output Functions

The following table 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 determines the value that the Analog Function Block will assign to the output data point.

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 the *FN_COMPARE Example* section.
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 the *FN_AND Example* section.
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 either state or value field is 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

Identifier	Value Assigned To The Output Data Point
	the analog function block are True. For an example of how you could use the FN_OR function in this way, see the *FN_OR Example* section.

8.3.2.2 Comparison Functions

The following table 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.

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	No comparison function is used. This is true if <UCPToutput Function> is set to FN_MAX, FN_MIN, FN_SUM, or FN_AVERAGE.

8.3.2.2.1 FN_AND Example

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**.

<UCPToutputFunction>: FN_AND
<UCPTcompFunction>: FN_GT

Because the output function is FN_AND, the comparisons made with all the input data points must return True 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 greater than the value of the <UCPTtrueThreshold> value (if 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.

The following table lists several case scenarios that show when these functions might would evaluate to True (100.0 1).

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

8.3.2.2.2 FN_OR Example

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**.

<UCPToutputFunction>: FN_OR
<UCPTcompFunction>: FN_LT

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.

The following table lists several case scenarios that show when these two functions might evaluate to True (100.0 1).

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	15	EMPTY	0.0 0
20	30	40	50	25	EMPTY	100.0 1
20	30	40	50	35	EMPTY	100.0 1

8.3.2.2.3 FN_COMPARE Example

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**.

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

Because the <UCPTmajorityValue> is set to 100, all comparisons made between the input and compare data points must return True 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.

The following table lists several case scenarios that show when these two functions might evaluate to True.

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

8.3.3 Using the Set Function on an Analog Functional Block

Use the *Set* function to create new Analog Functional Blocks, or to overwrite the configuration of existing Analog Functional Blocks. The Analog Functional Blocks to be created or written to are signified by a list of <Item> elements in the input parameters supplied to the function. The properties you must define within each <Item> element are the same, whether you are creating a new Analog Functional Block or modifying an existing Analog Functional Block. The previous section, *Using the Get Function on an Analog Functional Block*, describes these properties.

Note: If you specify an Analog Functional Block with the <UCPTname> element, the *Set* function deletes the specified Analog Functional Block before the specified parameters are set. If the <UCPTname> element is not specified, a new Analog Functional Block is created.

When modifying an existing Analog Functional Block, any optional properties omitted from the *Set* Request, such as the input point, compare point, or output data points, will be erased. Old values will not be preserved, so you must fill in every property when writing to an Analog Functional Block, even if you are not changing all of the values.

The first invocation of the *Set* function will generate the #8000010128000000[4].UFPTanalogFunctionBlock.xml file in the root/config/network/<network>/<channel>/iLONApp ||<device> directory of the SmartServer, if the file does not already exist.

When creating or modifying an Analog Functional Block with the *Set* function, you may want to use output from the *Get* function as the basis for your input. 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 example below uses the *Set* function to create an Analog Function Block that calculates the maximum value of the value field of the **nviClaValue_1** and **nviClaValue_2** data points on the **Digital Output 1** and **Digital Output 2** functional blocks on the SmartServer, and stores that value in the value field of the **nvoClsValue_1** data point on the **Digital Input 1** functional block on the SmartServer.

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTanalogFunctionBlock_Cfg">
        <UCPTname>Net/LON/iLON App/Analog Fn Block[2]</UCPTname>
        <UCPTdescription>enter an optional description</UCPTdescription>
        <DataPoint dpType="nvoDropOut" discrim="dir_out">
            <UCPTname>Net/LON/iLON App/Analog Fn Block[2]/nvoAfbDropOut[2]</UCPTname>
        </DataPoint>
        <DataPoint xsi:type="UFPTanalogFunctionBlock_DpRef" dpType="Input" discrim="dir_in">
            <UCPTname>Net/LON/iLON App/Digital Output 1/nviClaValue_1</UCPTname>
            <UCPTfieldName>value</UCPTfieldName>
            <UCPTpollRate>900</UCPTpollRate>
        </DataPoint>
        <DataPoint xsi:type="UFPTanalogFunctionBlock_DpRef" dpType="Input" discrim="dir_in">
            <UCPTname>Net/LON/iLON App/Digital Output 2/nviClaValue_2</UCPTname>
            <UCPTfieldName>value</UCPTfieldName>
            <UCPTpollRate>900</UCPTpollRate>
        </DataPoint>
        <DataPoint xsi:type="UFPTanalogFunctionBlock_DpRef" dpType="Output">
            <UCPTname>Net/LON/iLON App/Digital Input 1/nvoClsValue_1</UCPTname>
            <UCPTfieldName>value</UCPTfieldName>
        </DataPoint>
        <UCPTcompFunction xsi:type="string" LonFormat="UCPTcompFunction">FN_NUL</UCPTcompFunction>
        <UCPTmajorityValue>100</UCPTmajorityValue>
        <UCPTtrueThreshold xsi:type="string" LonFormat="UNVT_float"/>
            <UCPToutputFunction xsi:type="string" LonFormat="UCPToutputFunction">FN_INVALID
        </UCPToutputFunction>
        <SCPTminRnge xsi:type="string" LonFormat="UNVT_float">0</SCPTminRnge>
        <SCPTmaxRnge xsi:type="string" LonFormat="UNVT_float">0</SCPTmaxRnge>
        <UCPTcalculationInterval>0</UCPTcalculationInterval>
        <SCPTovrBehave xsi:type="string" LonFormat="SCPTovrBehave">OV_RETAIN</SCPTovrBehave>
        <UCPTpollOnResetDelay>0</UCPTpollOnResetDelay>
```

```
    </Item>
  </iLonItem>
</Set>
```

8.3.4 Using the Delete Function on an Analog Function Block

You can use the *Delete* function to delete an Analog Functional Block. To delete an Analog Functional Block, you provide an <Item> element with a UFPTanalogFunctionBlock_Cfg type that includes the <UCPTname> property of the analog functional block to be deleted. The following code sample demonstrates how to use the *Delete* function to delete an Analog Functional Block:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <Item xsi:type="UFPTanalogFunctionBlock_Cfg">
            <UCPTname> Net/LON/iLON App/Analog Functional Block[2]</UCPTname>
        </Item>
    </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/LON/iLON App/Analog Functional Block[2]</UCPTname>
    </Item>
 </iLonItem>
</DeleteResponse>
```

9 Scheduler

You can use the 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 Scheduler you create. These data points will be updated to specific values on the dates and times that the Scheduler is active. The dates and times that the Scheduler is active, as well as the values the Scheduler will assign to the data points, are user-defined. This section provides an overview of how the Scheduler application 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 each 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 a 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 weekdays Monday through Friday, and leaves the data point set to *off* on Saturday and Sunday.

Date-Based Schedules

You can create date-based exceptions for each 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 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 a Scheduler. The date-based exceptions must be created with the Calendar application. This is described in Chapter 10.

Sunrise and Sundown Events

You can schedule events to occur at sunrise or sundown (or a configure period of time before or after sunrise or sundown) in a day-based or date-based schedule.

#UNLOCK and #LOCK Events

You can schedule #UNLOCK events that enable lower-priority events to write updated values to the data point in the Scheduler, and you can schedule #LOCK events that prevent lower-priority events from updating the data points. You can schedule #UNLOCK and #LOCK events in a day-based or date-based schedule.

Data Points

The Scheduler application allows the integrator to dynamically select data points of any standard or user-defined network variable type to be updated by a 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 Scheduler application.

Refreshing Exceptions

You can use the Calendar application to create the exception points that define the date-based schedules for your Schedulers. Chapter 10 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 **nviDateEvent** point.

All Schedulers on the SmartServer read the exception list from the local NodeObject internally (without 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 **nvoEcDateEvent** output of the Calendar, and the **nviDateEvent** input of the NodeObject.

After a restart, the Scheduler recalculates the last Scheduler operation. It also sets the data point **nvoDateResync** to "100.0 1", which updates the SmartServer's exception list. You can set the value of nvoDateResync to "100.0 1" and then return it back to "0.0 0" with the *Write* function at any time to refresh the exception list manually. The data point **nviEcResync** of the Calendar will be internally bound to **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.

9.1 Overview of the Scheduler XML File

The #8000010128000000[4].UFPTscheduler.xml file stores the configuration of the Schedulers that you have added to the SmartServer. You can create new Schedulers using the *Set* function, or by manually editing the #8000010128000000[4].UFPTscheduler.xml file, and rebooting the SmartServer.

You can create up to 40 Schedulers per SmartServer. You can add more than 40 Schedulers if you load the dynamic v40 XIF on your SmartServer and you operate your SmartServer in Standalone mode. Note that using the v40 XIF with the SmartServer operating in LNS mode (**LNS Auto** or **LNS Manual**) is not supported.

The following represents a sample #8000010128000000[4].UFPTscheduler.xml file for a SmartServer with one Scheduler.

```
<Item xsi:type="UFPTscheduler_Cfg" >
        <UCPTname>Net/LON/iLON App/Scheduler[0]</UCPTname>
        <UCPTannotation>8000010128000000[4].UFPTscheduler</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-03T14:50:04.680-08:00</UCPTlastUpdate>
        <UCPTuri>#8000010128000000[4].UFPTscheduler_Cfg.htm</UCPTuri>
        <DataPoint dpType="nviEnable" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Scheduler[0]/nviEsEnable[0]</UCPTname>
        </DataPoint>
        <DataPoint xsi:type="UFPTscheduler_DpRef" dpType="Output" discrim="dir_out" >
            <UCPTname>Net/LON/iLON App/Digital Output 1/nviClaValue_1</UCPTname>
            <SCPTdelayTime>0</SCPTdelayTime>
        </DataPoint>
        <DataPoint xsi:type="UFPTscheduler_DpRef" dpType="Output" discrim="dir_out" >
            <UCPTname>Net/LON/iLON App/Digital Output 2/nviClaValue_2</UCPTname>
            <SCPTdelayTime>0</SCPTdelayTime>
        </DataPoint>
        <ScheduleEffectivePeriod>
            <StartDate>2000-01-01</StartDate>
            <EndDate>2037-12-31</EndDate>
        </ScheduleEffectivePeriod>
        <DayBased>
            <UCPTindex>0</UCPTindex>
            <UCPTdescription>Weekday</UCPTdescription>
            <UCPTpriority>255</UCPTpriority>
            <Event>
                <UCPTindex>0</UCPTindex>
                <UCPTtime>00:00:00</UCPTtime>
                <UCPTeventType LonFormat="UCPTeventType">ET_LOCK</UCPTeventType>
            </Event>
            <Event>
                <UCPTindex>1</UCPTindex>
                <UCPTtime>09:00:00</UCPTtime>
                <UCPTvalue LonFormat="">ON</UCPTvalue>
                <UCPTvalue LonFormat="UCPTvalueDef">ON</UCPTvalue>
                <UCPTtimeDirection LonFormat="UCPTtimeDirection">TD_POSITIVE</UCPTtimeDirection>
                <UCPTeventType LonFormat="UCPTeventType">ET_NUL</UCPTeventType>
            </Event>
            <Weekdays>
                <UCPTsunday>0</UCPTsunday>
                <UCPTmonday>1</UCPTmonday>
                <UCPTtuesday>1</UCPTtuesday>
```

```
            <UCPTwednesday>1</UCPTwednesday>
            <UCPTthursday>1</UCPTthursday>
            <UCPTfriday>1</UCPTfriday>
            <UCPTsaturday>0</UCPTsaturday>
        </Weekdays>
    </DayBased>
    <DayBased>
        <UCPTindex>1</UCPTindex>
        <UCPTdescription>Weekend</UCPTdescription>
        <UCPTpriority>255</UCPTpriority>
        <Event>
            <UCPTindex>0</UCPTindex>
            <UCPTtime>00:00:00</UCPTtime>
            <UCPTeventType LonFormat="UCPTeventType">ET_LOCK</UCPTeventType>
        </Event>
        <Weekdays>
            <UCPTsunday>1</UCPTsunday>
            <UCPTmonday>0</UCPTmonday>
            <UCPTtuesday>0</UCPTtuesday>
            <UCPTwednesday>0</UCPTwednesday>
            <UCPTthursday>0</UCPTthursday>
            <UCPTfriday>0</UCPTfriday>
            <UCPTsaturday>1</UCPTsaturday>
        </Weekdays>
    </DayBased>
</Item>
```

9.2 Creating and Modifying the Scheduler XML File

You can create and manage the #8000010128000000[4].UFPTscheduler.xml file with the *Set* SOAP function. The following section, *Scheduler SOAP Interface*, describes how to use *Set* and the other SOAP functions provided for the Scheduler application.

Alternatively, you can create and manage the #8000010128000000[4].UFPTscheduler.xml file manually with an XML editor and download it to the SmartServer via FTP. Echelon does not recommend this, as the SmartServer will require a reboot to read the configuration of the downloaded file. Additionally, the SmartServer 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 Scheduler's configuration.

9.3 Scheduler SOAP Interface

You can use the SOAP interface to perform the following functions on a Scheduler application:

Function	Description
List	Generate a list of Schedulers that you have added to the SmartServer.
Get	Retrieve the configuration of any Scheduler that you have added to the SmartServer.
Read	Retrieve the events scheduled in any Scheduler that you have added to the SmartServer.
Set	Create a new Scheduler, or overwrite the configuration of an existing Scheduler.
Delete	Delete a Scheduler.

Note: Section 21.1.3, *Creating a Scheduler and Calendar in Visual C# .NET*, includes a C# programming example demonstrating how to use the Scheduler SOAP interface to create and read a data logger. Section 21.2.3, *Creating a Scheduler and Calendar in Visual Basic.NET*, includes a Visual Basic example demonstrating how to do this.

9.3.1 Using the List Function on a Scheduler

You can use the *List* function to retrieve a list of the Schedulers that you have added to the SmartServer. The List function takes an <iLonItem> element that includes an xSelect statement querying items of a UFPTscheduler_Cfg type as its input, as shown in the example below. The *List* function returns an <Item> element for each Scheduler that you have added to the SmartServer. The next section describes the properties included in each of these elements.

You could use the list of <Item> elements returned by this function as input for the *Get* function. The *Get* function would then return the configuration of each Scheduler included in the list.

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <xSelect>//Item[@xsi:type=" UFPTscheduler_Cfg"]</xSelect>
    </iLonItem>
</List>
```

Response

```
<ListResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>Net/LON/iLON App/Scheduler[0]</UCPTname>
        <UCPTannotation>#8000010128000000[4].UFPTscheduler;xsi:type="LON_Fb_Cfg”;
        </UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
  </iLonItem>
 </ListResponse>
```

9.3.2 Using the Get Function a Scheduler

You can use the *Get* function to retrieve the configuration of any Scheduler that you have added to the SmartServer. You must reference the Scheduler whose configuration is to be returned by its <UCPTname> in the input you supply to the function, as shown in the example below.

Request

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>Net/LON/iLON App/Scheduler[0]</UCPTname>
        </Item>
  </iLonItem>
</Get>
```

Response

```
<GetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="UFPTscheduler_Cfg" >
        <UCPTname>Net/LON/iLON App/Scheduler[0]</UCPTname>
        <UCPTannotation>8000010128000000[4].UFPTscheduler</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-03T14:50:04.680-08:00</UCPTlastUpdate>
        <UCPTuri>#8000010128000000[4].UFPTscheduler_Cfg.htm</UCPTuri>
        <DataPoint xsi:type="UFPTscheduler_DpRef" dpType="Output" discrim="dir_out" >
            <UCPTname>Net/LON/iLON App/Digital Output 1/nviClaValue_1</UCPTname>
            <SCPTdelayTime>0</SCPTdelayTime>
        </DataPoint>
        <DataPoint xsi:type="UFPTscheduler_DpRef" dpType="Output" discrim="dir_out" >
```

```
            <UCPTname>Net/LON/iLON App/Digital Output 2/nviClaValue_2</UCPTname>
            <SCPTdelayTime>0</SCPTdelayTime>
        </DataPoint>
        <ScheduleEffectivePeriod>
            <StartDate>2000-01-01</StartDate>
            <EndDate>2037-12-31</EndDate>
        </ScheduleEffectivePeriod>
        <DayBased>
            <UCPTindex>0</UCPTindex>
            <UCPTdescription>Weekday</UCPTdescription>
            <UCPTpriority>255</UCPTpriority>
            <Event>
                <UCPTindex>0</UCPTindex>
                <UCPTtime>00:00:00</UCPTtime>
                <UCPTeventType LonFormat="UCPTeventType">ET_LOCK</UCPTeventType>
            </Event>
            <Event>
                <UCPTindex>1</UCPTindex>
                <UCPTtime>09:00:00</UCPTtime>
                <UCPTvalue LonFormat="">ON</UCPTvalue>
                <UCPTvalue LonFormat="UCPTvalueDef">ON</UCPTvalue>
                <UCPTtimeDirection LonFormat="UCPTtimeDirection">TD_POSITIVE</UCPTtimeDirection>
                <UCPTeventType LonFormat="UCPTeventType">ET_NUL</UCPTeventType>
            </Event>
            <Weekdays>
                <UCPTsunday>0</UCPTsunday>
                <UCPTmonday>1</UCPTmonday>
                <UCPTtuesday>1</UCPTtuesday>
                <UCPTwednesday>1</UCPTwednesday>
                <UCPTthursday>1</UCPTthursday>
                <UCPTfriday>1</UCPTfriday>
                <UCPTsaturday>0</UCPTsaturday>
            </Weekdays>
        </DayBased>
        <DayBased>
            <UCPTindex>1</UCPTindex>
            <UCPTdescription>Weekend</UCPTdescription>
            <UCPTpriority>255</UCPTpriority>
            <Event>
                <UCPTindex>0</UCPTindex>
                <UCPTtime>00:00:00</UCPTtime>
                <UCPTeventType LonFormat="UCPTeventType">ET_LOCK</UCPTeventType>
            </Event>
            <Weekdays>
                <UCPTsunday>1</UCPTsunday>
                <UCPTmonday>0</UCPTmonday>
                <UCPTtuesday>0</UCPTtuesday>
                <UCPTwednesday>0</UCPTwednesday>
                <UCPTthursday>0</UCPTthursday>
                <UCPTfriday>0</UCPTfriday>
                <UCPTsaturday>1</UCPTsaturday>
            </Weekdays>
        </DayBased>
        <DateBased>
            <UCPTindex>0</UCPTindex>
            <UCPTpriority>250</UCPTpriority>
                <Event>
                    <UCPTindex>0</UCPTindex>
                    <UCPTtime>04:00:00</UCPTtime>
                    <UCPTvalue LonFormat="">OFF</UCPTvalue>
                    <UCPTvalue LonFormat="UCPTvalueDef">OFF</UCPTvalue>
                    <UCPTtimeDirection LonFormat="UCPTtimeDirection">TD_POSITIVE</UCPTtimeDirection>
                    <UCPTeventType LonFormat="UCPTeventType">ET_NUL</UCPTeventType>
                </Event>
                <Exception>
                    <UCPTindex>0</UCPTindex>
                    <UCPTexceptionName>2008-03-03</UCPTexceptionName>
                </Exception>
        </DateBased>
    </Item>
</iLonItem>
```

```
</GetResponse>
```

The function returns an <Item> element for each Scheduler referenced in the input parameters supplied to the function. The properties included in each element are initially defined when the Scheduler is created. You can write to these properties with the *Set* function. The following table describes these properties.

Property	Description
`<UCPTname>`	The name of the Scheduler in the following format: <*network*/*channel*/*device*/*functional block*>.
`<UCPTannotation>`	The program ID and functional profile template used by the Scheduler. This property is always `8000010128000000[4].UFPTscheduler`
`<UCPThidden>`	A flag indicating whether the Scheduler functional block is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **`0`** – shown **`1`** – hidden
`<UCPTitemStatus>`	This property only appears if the data logger is not synchronized with an LNS network database or it has been deleted. In this case, it has the following values: **`IS_NOTSYNCED`** **`IS_DELETED`**
`<UCPTlastUpdate>`	A timestamp indicating the last time the configuration of the Scheduler was updated. This timestamp uses the following format: YYYY-MM-DD**T**HH:MM:SS**Z**
`<UCPTuri>`	The name of the file containing the configuration web page for the Scheduler on the SmartServer flash disk, absolute or relative to /web/user/echelon. This property is `#8000010128000000[4].UFPTscheduler _Cfg.htm` by default.
`<DataPoint>` `Input`	For each <Data 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 a 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**

Property	Description
	will be decreased with every heartbeat.
<ScheduleEffectivePeriod>	The <ScheduleEffectivePeriod> element contains two properties that define the dates that the Scheduler applies to. The <StartDate> property defines the start date, and the <EndDate> property defines the end date. You must fill each property in using the following format: YYYY-MM-DD If the start date is undefined (0000-00-00), it means any date up to and including the end date. If the end date is undefined, it means any date from the start date. If both are undefined, it means the Scheduler is always active. The default value for both properties is 0000-00-00.
<DayBased>	Each 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 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*.
<DateBased>	Each Scheduler can have one date-based schedule. You will reference the schedule exceptions created with the 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*.

9.3.2.1 Creating a Day-Based Schedule

The following code demonstrates the structure of the <DayBased> element. The subsequent table lists and describes the properties that should be defined within each <DayBased> element.

```
<DayBased>
    <UCPTindex>0</UCPTindex>
    <UCPTdescription>Weekday</UCPTdescription>
    <UCPTpriority>255</UCPTpriority>
    <Event>
        <UCPTindex>0</UCPTindex>
        <UCPTtime>00:00:00</UCPTtime>
        <UCPTeventType LonFormat="UCPTeventType">ET_LOCK</UCPTeventType>
    </Event>
    <Event>
        <UCPTindex>1</UCPTindex>
        <UCPTtime>09:00:00</UCPTtime>
        <UCPTvalue LonFormat="">ON</UCPTvalue>
        <UCPTvalue LonFormat="UCPTvalueDef">ON</UCPTvalue>
        <UCPTtimeDirection LonFormat="UCPTtimeDirection">TD_POSITIVE</UCPTtimeDirection>
        <UCPTeventType LonFormat="UCPTeventType">ET_NUL</UCPTeventType>
    </Event>
```

```
    <Event>
        <UCPTindex>2</UCPTindex>
        <UCPTtime>10:00:00</UCPTtime>
        <UCPTvalue LonFormat="">0.0 0</UCPTvalue>
        <UCPTvalue LonFormat="UCPTvalueDef">0.0 0</UCPTvalue>
        <UCPTtimeDirection LonFormat="UCPTtimeDirection">TD_POSITIVE</UCPTtimeDirection>
        <UCPTeventType LonFormat="UCPTeventType">ET_NUL</UCPTeventType>
    </Event>
    <Weekdays>
        <UCPTsunday>0</UCPTsunday>
        <UCPTmonday>1</UCPTmonday>
        <UCPTtuesday>1</UCPTtuesday>
        <UCPTwednesday>1</UCPTwednesday>
        <UCPTthursday>1</UCPTthursday>
        <UCPTfriday>1</UCPTfriday>
        <UCPTsaturday>0</UCPTsaturday>
    </Weekdays>
</DayBased>
```

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 227 characters long.
<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 Scheduler attempts to update that data point. If it is not, the data point will not be updated successfully.
<Event>	The update events for each day-based schedule are signified by a list of <Event> elements. Each update event will be used on the days that this day-based schedule is active. See *Creating Events* for more information. Note that each <Day Based> element has an <Event> item that specifies a LOCK event to occur at midnight (00:00).

<Weekdays>	The <Weekdays> element contains seven properties, one for each day of the week. If you set the property for a day to 1, this day-based schedule will be active on that day. Otherwise, it will be inactive. For example, to create a day-based schedule that is active on Monday and Tuesday, use the following <Weekdays> element:
	`<Weekdays>`
	`<UCPTsunday>0</UCPTsunday>`
	`<UCPTmonday>1</UCPTmonday>`
	`<UCPTtuesday>1</UCPTtuesday>`
	`<UCPTwednesday>0</UCPTwednesday>`
	`<UCPTthursday>0</UCPTthursday>`
	`<UCPTfriday>0</UCPTfriday>`
	`<UCPTsaturday>0</UCPTsaturday>`
	`</Weekdays>`

9.3.2.2 Creating a Date-Based Schedule

The following code demonstrates the structure of the <DateBased> element. This example includes two <DateBased> elements that have different <Exception> properties. The first <DateBased> element is a one-time exception that occurs on a specific date. The second <DateBased> element is an

exception that occurs over a specific range of dates. The actual date or range of dates in which these exceptions occur are stored in the #8000010128000000[4].UFPTcalendar.xml file.

The subsequent table lists and describes the properties that should be defined within each <DateBased> element.

```
<DateBased>
    <UCPTindex>0</UCPTindex>
    <UCPTpriority>250</UCPTpriority>
    <Event>
        <UCPTindex>0</UCPTindex>
        <UCPTtime>04:00:00</UCPTtime>
        <UCPTvalue LonFormat="">OFF</UCPTvalue>
        <UCPTvalue LonFormat="UCPTvalueDef">OFF</UCPTvalue>
        <UCPTtimeDirection LonFormat="UCPTtimeDirection">TD_POSITIVE</UCPTtimeDirection>
        <UCPTeventType LonFormat="UCPTeventType">ET_NUL</UCPTeventType>
    </Event>
    <Exception>
        <UCPTindex>0</UCPTindex>
        <UCPTexceptionName>2008-03-03</UCPTexceptionName>
    </Exception>
</DateBased>
<DateBased>
    <UCPTindex>1</UCPTindex>
    <UCPTpriority>250</UCPTpriority>
    <Event>
        <UCPTindex>0</UCPTindex>
        <UCPTtime>00:00:00</UCPTtime>
        <UCPTeventType LonFormat="UCPTeventType">ET_LOCK</UCPTeventType>
    </Event>
    <Event>
        <UCPTindex>1</UCPTindex>
        <UCPTtime>00:15:00</UCPTtime>
        <UCPTvalue LonFormat="">OFF</UCPTvalue>
        <UCPTvalue LonFormat="UCPTvalueDef">OFF</UCPTvalue>
        <UCPTtimeDirection LonFormat="UCPTtimeDirection">TD_NEGATIVE</UCPTtimeDirection>
        <UCPTbaseTimePath>../../DataPoint[UCPTnickName="Sunrise"]</UCPTbaseTimePath>
        <UCPTeventType LonFormat="UCPTeventType">ET_NUL</UCPTeventType>
    </Event>
    <Exception>
        <UCPTindex>0</UCPTindex>
        <UCPTexceptionName>SunriseSchedule</UCPTexceptionName>
  </Exception>
</DateBased>
```

Property	Description
<UCPTindex>	The index number for the date-based schedule.
<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 Scheduler attempts to update the data point. If it is not, the data point will not be updated successfully.
<Event>	The update events for each date-based schedule are signified by a list of <Event> elements. Each update event will be used on the days that this date-based schedule is active. See *Creating Events* for more information.

Property	Description
<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 Calendar application. For more information on this, see Chapter 10, *Calendar*.

9.3.2.3 Creating Events

The update events for each day-based or date-based schedule are signified by a list of <Event> elements. Each update event will be used on the days that this day-based or date-based schedule is active. Note that there is no limit on the number of events supported by the Scheduler application.

```
<Event>
    <UCPTindex>1</UCPTindex>
    <UCPTtime>00:15:00</UCPTtime>
    <UCPTvalue LonFormat="">OFF</UCPTvalue>
    <UCPTvalue LonFormat="UCPTvalueDef">OFF</UCPTvalue>
    <UCPTtimeDirection LonFormat="UCPTtimeDirection">TD_NEGATIVE</UCPTtimeDirection>
    <UCPTbaseTimePath>../../DataPoint[UCPTnickName="Sunrise"]</UCPTbaseTimePath>
    <UCPTeventType LonFormat="UCPTeventType">ET_NUL</UCPTeventType>
  </Event>
```

Each <Event> element contains the following properties:

Property	Description
<UCPTindex>	The index number for the event.
<UCPTtime>	If you are creating a time-of-day event, this is the local time that defines the time of day when the data points selected for your Scheduler will be updated to the value specified for the <UCPTValue> property. This time must be entered in 24-hour format, e.g. 15:30:00 represents 3:30:00 PM. If you are creating an event that is to occur sometime before or after sunrise or sundown, enter the offset to be applied to the calculated sunrise or sundown time. The offset must be entered in 24-hour format with a valid range between 00:01:00 and 23:59:00.
<UCPTvalue>	The value to be written to the data point at the time specified in the <UCPTtime> property. You can specify one or more <UCPTvalue> elements with an empty LonFormat attribute, or with a LonFormat attribute that is set to "UCPTvalueDef", and you can set <UCPTvalue> to a value string such as "100.0 1" or a value definition (preset) such as "ON". The last <UCPTvalue> property defined is the one that is used. For example, you can define a value string for a **SNVT_switch** data point using either of the following statements: <UCPTvalue LonFormat="">100.0 1</UCPTvalue> <UCPTvalue LonFormat="UCPTvalueDef">100.0 1</UCPTvalue> Alternatively, you can define a value definition (preset) for a

Property	Description

	SNVT_switch data point using either of the following statements:
	`<UCPTvalue LonFormat="">ON</UCPTvalue>`
	`<UCPTvalue LonFormat="UCPTvalueDef">ON</UCPTvalue>`

<UCPTtimeDirection>	If you are creating an event that is to occur sometime before or after sunrise or sundown, specify in a LonFormat attribute whether the offset specified in the <UCPTtime> property is to be added to or subtracted from the calculated sunrise or sundown time. This property can have one of the following values: • **TD_POSITIVE**: The offset specified in the <UCPTtime> property is to be added to the calculated sunrise or sundown time. For all time-of-day events, you must specify TD_POSITIVE. • **TD_NEGATIVE**: The offset specified in the <UCPTtime> property is to be subtracted from the calculated sunrise or sundown time. For example, to specify that an event is to occur 15 minutes before sunrise, you would use the following code: `<UCPTtimeDirection LonFormat="UCPTtimeDirection">TD_NEGATIVE </UCPTtimeDirection>` **Note**: You can omit this property if you are creating a time-of-day event.

<UCPTbaseTimePath>	If you are creating a sunrise or sundown event, specify a reference to the data point information which is used as base time for this event. The reference is written as xPath statement relative to the actual element.
	To create a sunrise event, you would use the following code:
	`<UCPTbaseTimePath>../../DataPoint[UCPTnickName="Sunrise"] </UCPTbaseTimePath>`
	To create a sundown event, you would use the following code:
	`<UCPTbaseTimePath>../../DataPoint[UCPTnickName="Sunset"] </UCPTbaseTimePath>`

<EventType>	You can use this element to create a #LOCK or #UNLOCK event. This element may have one of the following values: • **ET_LOCK:** Locks out lower priority events and prevents them from writing to the data points in the Scheduler. Applications with lower priorities can only override a data point value once the Scheduler has executed an #UNLOCK event. • **ET_UNLOCK:** Enables lower priority events to write to the data points in the Scheduler. • **ET_NUL:** This is the default, indicating that the event is based on the time-of-day, or sunrise or sundown. You need to specify this value in a LonFormat attribute that has the following value: "UCPTeventType". For example, to create a #LOCK event, you would use the following code: `<UCPTeventType LonFormat="UCPTeventType">ET_LOCK </UCPTeventType>`

9.3.3 Using the Read Function on a Scheduler

You can use the *Read* function to retrieve the events scheduled in any Scheduler that you have added to the SmartServer. You can filter events using an xSelect statement and specifying one or more Scheduler items. You can use the following filters in an xSelect statement when using the *Read* function on a Scheduler:

UCPTlastUpdate	When an event occurs. You can compare the specified start and stop times using equal, less (or equal), great (or equal). UCPTstop is set to time of last shown event. You need to check whether UCPTstop differs from UCPTlastUpdate to make the next request start from the UCPTstop time.
UCPTeventFilter	This property may have one of following values: • **EF_NUL:** Show all events. This is the same as not specifying and UCPTeventFilter. • **EF_DAY:** Show only first event scheduled for each day. • **EF_SCHEDULE**: Show only one event for each scheduler.
position()	Position of the event in previously selected list. You can use this filter to limit the number of events in a result. You can compare the number of events to be returned using equal, less (or equal), or great (or equal). You can request a maximum of 2,500 events; however, more than 2,500 events may be returned. **Note**: The last instance of an event before the specified start date is always returned (if it exists), and it is not counted by the position() constraint.

You can include one item of type UFPTscheduler_Calendar_Request_Data, which causes a Read function on a Calendar item. This information is used to calculate date-based events. You can also include an item of type UFPTscheduler_RealTimeClock_Request_Data, which defines start point for relative times for each day. By default, all relative times are calculated from 0:00:00. If calendar and/or real time clock data is specified, the data in the response is bound to given start and stop dates.

Request

```
<Read xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect> //Item[@xsi:type="UFPTscheduler_Data"]
                  [UCPTlastUpdate&gt;= "2008-03-01T00:00:00.000" and
                   UCPTlastUpdate&lt;= "2008-04-01T00:00:00.000"]
                  [UCPTeventFilter="EF_NUL"]
                  [position()&lt;500]
    </xSelect>
    <Item xsi:type="UFPTscheduler_Calendar_Request_Data">
        <UCPTname>Net/LON/iLON App/Calendar</UCPTname>
    </Item>
    <Item xsi:type="UFPTscheduler_RealTimeClock_Request_Data">
        <UCPTname>Net/LON/iLON App/Real Time Clock</UCPTname>
    </Item>
    <Item>
        <UCPTname>Net/LON/iLON App/Scheduler[0]__TEMP_OBJECT</UCPTname>
    </Item>
  </iLonItem>
</Read>
```

Response

```
<ReadResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
```

```
<Item xsi:type="UFPTscheduler_Meta_Data" >
    <UCPTname>Net/LON/iLON App/Scheduler[0]</UCPTname>
    <UCPTstart>2008-03-01T00:00:00.000-08:00</UCPTstart>
    <UCPTstop>2008-04-01T00:01:00.000-07:00</UCPTstop>
</Item>
<Item xsi:type="UFPTscheduler_Data" >
    <UCPTname/>
    <UCPTlastUpdate>2008-03-01T00:00:00.000-08:00</UCPTlastUpdate>
    <UCPTdescription>Weekend</UCPTdescription>
    <UCPTpriority>255</UCPTpriority>
    <UCPTmetaDataPath>//*[@xsi:type="UFPTscheduler_Meta_Data"][UCPTname="Net/LON/iLON
    App/Scheduler[0]"]</UCPTmetaDataPath>
    <UCPTeventType LonFormat="UCPTeventType">ET_LOCK</UCPTeventType>
    <UCPTeventSource>DayBased[UCPTindex=1]/Event[UCPTindex=0]</UCPTeventSource>
    <UCPTeventInfo LonFormat="UCPTeventInfo">EI_BEGIN</UCPTeventInfo>
</Item>
<Item xsi:type="UFPTscheduler_Data" >
    <UCPTname/>
    <UCPTlastUpdate>2008-03-02T00:00:00.000-08:00</UCPTlastUpdate>
    <UCPTdescription>Weekend</UCPTdescription>
    <UCPTpriority>255</UCPTpriority>
    <UCPTmetaDataPath>//*[@xsi:type="UFPTscheduler_Meta_Data"][UCPTname="Net/LON/iLON
    App/Scheduler[0]"]</UCPTmetaDataPath>
    <UCPTeventType LonFormat="UCPTeventType">ET_LOCK</UCPTeventType>
    <UCPTeventSource>DayBased[UCPTindex=1]/Event[UCPTindex=0]</UCPTeventSource>
    <UCPTeventInfo LonFormat="UCPTeventInfo">EI_END</UCPTeventInfo>
</Item>
<Item xsi:type="UFPTscheduler_Data" >
       <UCPTname/>
       <UCPTlastUpdate>2008-03-15T00:00:00.000-07:00</UCPTlastUpdate>
       <UCPTdescription/>
       <UCPTpriority>250</UCPTpriority>
       <UCPTmetaDataPath>//*[@xsi:type="UFPTscheduler_Meta_Data"][UCPTname="Net/LON/iLON
       App/Scheduler[0]"]</UCPTmetaDataPath>
       <UCPTeventType LonFormat="UCPTeventType">ET_LOCK</UCPTeventType>
       <UCPTeventSource>DateBased[UCPTindex=1]/(Event[UCPTindex=0] |
       Exception[UCPTindex=0])</UCPTeventSource>
       <UCPTexceptionName>WeekendShutdown</UCPTexceptionName>
       <UCPTeventInfo LonFormat="UCPTeventInfo">EI_BEGIN</UCPTeventInfo>
    </Item>
  </iLonItem>
</ReadResponse>
```

The *Read* function returns the following properties for each day or date read from the specified Scheduler:

<UCPTname>	The name of the scheduler from which events were read in the following format: <*network*/*channel*/*device*/*functional block*>.
<UCPTstart> <UCPTstop>	Timestamps indicating the times of the first and last dates of the events returned by the *Read* function.
<UCPTlastUpdate>	A timestamp indicating the time that the last event occurred.
<UCPTdescription>	The name of the day-based schedule in which the event is scheduled. This property is empty for date-based events.
<UCPTpriority>	The priority assigned the schedule, from 0 (highest priority) to 255 (lowest priority).
<UCPTmetaDataPath>	The queried data type and the UCPT name of the queried Scheduler.

<EventType>	This element may have one of the following values: • **ET_LOCK:** A #LOCK event. • **ET_UNLOCK:** An #UNLOCK event. • **ET_NUL:** Event is based on the time-of-day, or sunrise or sundown.
<EventSource>	This element indicates whether the event is in a day-based schedule or a date-based exception schedule. This property may have one of the following values: • **DayBased:** Event is scheduled in a daily schedule. • **DateBased:** Event is scheduled in an exception schedule. • This property includes the UCPTindex of the DayEvent or DateEvent and the UCPTindex of the event itself.
<UCPTexceptionName>	The name of the date-based exception schedule in which the event is scheduled. This property is not included for day-based events.
<UCPTeventInfo>	Indicates how the day or date in which the event occurs relates to the specified range of days or dates in which that event is scheduled. This property may have one of the following values: • **EI_BEGIN:** First day in the range of days or dates. • **EI_END:** Last day in the range of days or dates. • **EI_CONTINUE:** A day in between the beginning and the end of the range of days or dates. • **EI_BEGIN_END:** Event occurs only on one day.

9.3.4 Using the Set Function on a Scheduler

You can use the *Set* function to create new Schedulers, or to overwrite the configuration of existing Schedulers. The Schedulers to be created or written to are signified by a list of <Item> elements in the input you supply to the function. The properties you must define within each <Item> element are the same, whether you are creating a new Scheduler or modifying an existing Scheduler. The previous section, *Using the Get Function on a Scheduler*, describes these properties.

Note: If you specify a Scheduler with the <UCPTname> element, the *Set* function deletes the specified Scheduler before the specified parameters are set. If the <UCPTname> element is not specified, a new Scheduler is created.

The first invocation of the *Set* function will generate the #8000010128000000[4].UFPTscheduler.xml file in the root/config/network/<network>/<channel>/iLONApp ||<device> directory of the SmartServer, if the file does not already exist.

When modifying an existing Scheduler, any optional properties omitted from the input will be erased. Old values will not be preserved, so you should fill in every property when writing to a Scheduler, even if you are not changing all of the values.

When creating or modifying a Scheduler with this function, you may want to use output from the *Get* function as the basis for your input. 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 following example updates the **nviClaValue_1 SNVT_switch** data point on the Digital Ouput 1 functional block on the SmartServer. It includes one <DayBased> items that sets the data point to ON at 05:00, OFF at 09:00, and ON again at 17:00, and it contains one <DateBased> sets the data point to OFF 15 minutes before sunrise and unlocks the data point at sundown so that the Day Based schedule can update the data point.

Request

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
 <iLonItem>
   <Item xsi:type="UFPTscheduler_Cfg">
    <UCPTname>Net/LON/iLON App/Scheduler[0]</UCPTname>
    <UCPTdescription>enter an optional description</UCPTdescription>
    <DataPoint xsi:type="UFPTscheduler_DpRef" dpType="Output" discrim="dir_out">
        <UCPTname>Net/LON/iLON App/Digital Output 1/nviClaValue_1</UCPTname>
        <SCPTdelayTime>0</SCPTdelayTime>
    </DataPoint>
    <DataPoint xsi:type="UFPTscheduler_BaseTime_DpRef" dpType="BaseTime">
        <UCPTname>Net/LON/iLON App/Real Time Clock/nvoSunrise</UCPTname>
        <UCPTnickName>Sunrise</UCPTnickName>
    </DataPoint>
    <DataPoint xsi:type="UFPTscheduler_BaseTime_DpRef" dpType="BaseTime">
        <UCPTname>Net/LON/iLON App/Real Time Clock/nvoSunset</UCPTname>
        <UCPTnickName>Sunset</UCPTnickName>
    </DataPoint>
    <ScheduleEffectivePeriod xsi:type="UFPTscheduler_CfgEffectivePeriod">
        <StartDate>2000-01-01</StartDate>
        <EndDate>2037-12-31</EndDate>
    </ScheduleEffectivePeriod>
    <DayBased>
        <UCPTindex>0</UCPTindex>
        <UCPTdescription>Weekday</UCPTdescription>
        <UCPTpriority>255</UCPTpriority>
        <Event>
            <UCPTindex>0</UCPTindex>
            <UCPTtime>00:00:00</UCPTtime>
            <UCPTeventType xsi:type="string" LonFormat="UCPTeventType">ET_LOCK</UCPTeventType>
        </Event>
        <Event>
            <UCPTindex>1</UCPTindex>
            <UCPTtime>05:00:00</UCPTtime>
            <UCPTvalue LonFormat="">ON</UCPTvalue>
            <UCPTvalue LonFormat="UCPTvalueDef">ON</UCPTvalue>
            <UCPTtimeDirection xsi:type="string" LonFormat="UCPTtimeDirection">TD_POSITIVE
            </UCPTtimeDirection>
            <UCPTeventType xsi:type="string" LonFormat="UCPTeventType">ET_NUL</UCPTeventType>
        </Event>
        <Event>
            <UCPTindex>2</UCPTindex>
            <UCPTtime>09:00:00</UCPTtime>
            <UCPTvalue LonFormat="">OFF</UCPTvalue>
            <UCPTvalue LonFormat="UCPTvalueDef">OFF</UCPTvalue>
            <UCPTtimeDirection xsi:type="string" LonFormat="UCPTtimeDirection">TD_POSITIVE
            </UCPTtimeDirection>
            <UCPTeventType xsi:type="string" LonFormat="UCPTeventType">ET_NUL</UCPTeventType>
        </Event>
        <Event>
            <UCPTindex>3</UCPTindex>
            <UCPTtime>17:00:00</UCPTtime>
            <UCPTvalue LonFormat="">ON</UCPTvalue>
            <UCPTvalue LonFormat="UCPTvalueDef">ON</UCPTvalue>
            <UCPTtimeDirection xsi:type="string" LonFormat="UCPTtimeDirection">TD_POSITIVE
            </UCPTtimeDirection>
            <UCPTeventType xsi:type="string" LonFormat="UCPTeventType">ET_NUL</UCPTeventType>
```

```
        </Event>
        <Weekdays>
            <UCPTsunday>0</UCPTsunday>
            <UCPTmonday>1</UCPTmonday>
            <UCPTtuesday>1</UCPTtuesday>
            <UCPTwednesday>1</UCPTwednesday>
            <UCPTthursday>1</UCPTthursday>
            <UCPTfriday>1</UCPTfriday>
            <UCPTsaturday>0</UCPTsaturday>
        </Weekdays>
    </DayBased>
    <DayBased>
        <UCPTindex>1</UCPTindex>
        <UCPTdescription>Weekend</UCPTdescription>
        <UCPTpriority>255</UCPTpriority>
        <Event>
            <UCPTindex>0</UCPTindex>
            <UCPTtime>00:00:00</UCPTtime>
            <UCPTeventType xsi:type="string" LonFormat="UCPTeventType">ET_LOCK</UCPTeventType>
        </Event>
        <Weekdays>
            <UCPTsunday>1</UCPTsunday>
            <UCPTmonday>0</UCPTmonday>
            <UCPTtuesday>0</UCPTtuesday>
            <UCPTwednesday>0</UCPTwednesday>
            <UCPTthursday>0</UCPTthursday>
            <UCPTfriday>0</UCPTfriday>
            <UCPTsaturday>1</UCPTsaturday>
        </Weekdays>
    </DayBased>
    <DateBased>
        <UCPTindex>0</UCPTindex>
        <UCPTpriority>250</UCPTpriority>
        <Event>
            <UCPTindex>0</UCPTindex>
            <UCPTtime>00:00:00</UCPTtime>
            <UCPTeventType xsi:type="string" LonFormat="UCPTeventType">ET_LOCK</UCPTeventType>
        </Event>
        <Event>
            <UCPTindex>1</UCPTindex>
            <UCPTtime>00:15:00</UCPTtime>
            <UCPTvalue LonFormat="">OFF</UCPTvalue>
            <UCPTvalue LonFormat="UCPTvalueDef">OFF</UCPTvalue>
            <UCPTtimeDirection xsi:type="string" LonFormat="UCPTtimeDirection">TD_NEGATIVE
            </UCPTtimeDirection>
            <UCPTbaseTimePath xsi:type="string">../../DataPoint[UCPTnickName="Sunrise"]
            </UCPTbaseTimePath>
            <UCPTeventType xsi:type="string" LonFormat="UCPTeventType">ET_NUL</UCPTeventType>
        </Event>
        <Event>
            <UCPTindex>2</UCPTindex>
            <UCPTtime>00:00:00</UCPTtime>
            <UCPTtimeDirection xsi:type="string" LonFormat="UCPTtimeDirection">TD_POSITIVE
            </UCPTtimeDirection>
            <UCPTbaseTimePath xsi:type="string">../../DataPoint[UCPTnickName="Sunset"]
            </UCPTbaseTimePath>
            <UCPTeventType xsi:type="string" LonFormat="UCPTeventType">ET_UNLOCK</UCPTeventType>
        </Event>
        <Exception>
            <UCPTindex>0</UCPTindex>
            <UCPTexceptionName>WeekdayException</UCPTexceptionName>
        </Exception>
    </DateBased>
  </Item>
 </iLonItem>
</Set
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
```

```
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/LON/iLON App/Scheduler[0]</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

9.3.5 Using the Delete Function on a Scheduler

You can use the *Delete* function to delete a Scheduler. To delete a Scheduler, you provide an <Item> element with a UFPTscheduler_Cfg type that includes the <UCPTname> property of the scheduler to be deleted. The following code sample demonstrates how to use the *Delete* function to delete a Scheduler:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <Item xsi:type="UFPTscheduler_Cfg">
            <UCPTname>Net/LON/iLON App/Scheduler[0]</UCPTname>
        </Item>
    </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/LON/iLON App/Scheduler[0]</UCPTname>
    </Item>
 </iLonItem>
</DeleteResponse>
```

10 Calendar

You can use the Calendar application to define the exceptions that you will reference when creating the date-based schedules for your Schedulers. Each exception you create represents a date, or a group of dates. When you reference an exception in a Scheduler, you will be able to assign the dates for that exception a unique schedule. This may be useful when creating a 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 Calendar application. Chapter 9, *Scheduler*, describes how to create a Scheduler and reference the exceptions you create.

You can create one-time exceptions, exceptions that occur over a specific range of dates, or recurring exceptions that occur over a range of dates in a specific pattern such as every weekday or every fourth Sunday of every month. The SmartServer supports one active Calendar, with no limit on the number of exceptions scheduled in the Calendar.

When a Scheduler references an exception point, the Calendar application supplies the dates an exception point references to the Node Object using the **nvoEcDateEvent** data point. The 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 Calendar are recalculated and copied to the **nvoEcDateEvent** data point as a series of updates. By default, the **nvoEcDateEvent** data point of the Calendar and the **nviDateEvent** data point of the Node Object are internally bound, so that no network traffic is generated. Thus, the update from the Calendar is passed to the local Node Object, and all the Schedulers will read the updated exception list from the local Node Object.

In this fashion, each Scheduler will always have up-to-date definitions of the exceptions it references. To force all exceptions to be recalculated and copied to the **nvoEcDateEvent** data point, you may update the **nviEcDateResync** data point (which will be internally bound to the **nvoDateResync** data point of the Node Object if no external binding is created) with a value of “100.0 1” and then return it back to “0.0 0”.

10.1 Overview of the Calendar XML File

The #8000010128000000[4].UFPTcalendar.xml file stores the configuration of the Calendars that you have added to the SmartServer. You can create multiple Calendars with an unlimited number of exceptions per SmartServer. However, the SmartServer supports only 1 active Calendar at a time.

The following represents a sample #8000010128000000[4].UFPTcalendar.xml file for a SmartServer with a Calendar that has an exception group named Holiday that recurs annually with exception schedules defined for Christmas and the Fourth of July.

```
<Item xsi:type="UFPTcalendar_Cfg" >
        <UCPTname>Net/LON/iLON App/Calendar</UCPTname>
        <UCPTannotation>#8000010128000000[4].UFPTcalendar;xsi:type=“LON_Fb_Cfg”</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-05T11:43:29.440-08:00</UCPTlastUpdate>
        <UCPTuri>#8000010128000000[4].UFPTcalendar_Cfg.htm</UCPTuri>
        <DataPoint dpType="nviDateResync" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Calendar/nviEcDateResync</UCPTname>
        </DataPoint>
        <DataPoint dpType="nvoDateEvent" discrim="dir_out" >
            <UCPTname>Net/LON/iLON App/Calendar/nvoEcDateEvent</UCPTname>
        </DataPoint>
        <ScheduleEffectivePeriod>
            <StartDate>2000-01-01</StartDate>
            <EndDate>2037-12-31</EndDate>
        </ScheduleEffectivePeriod>
        <Exception>
            <UCPTindex>4</UCPTindex>
            <UCPTexceptionName>Holiday</UCPTexceptionName>
```

```
            <UCPTaliasName>Holiday</UCPTaliasName>
            <UCPTtemporary>0</UCPTtemporary>
            <Schedule>
                <StartDate>
                    <UCPTdate>2008-03-12</UCPTdate>
                    <UCPTyearMask LonFormat="UCPTyearMask">DW_NUL</UCPTyearMask>
                    <UCPTmonthMask LonFormat="UCPTmonthMask">DW_NUL</UCPTmonthMask>
                    <UCPTdayMask LonFormat="UCPTdayMask">DW_NUL</UCPTdayMask>
                </StartDate>
                <EndDate>
                    <UCPTdate>2015-12-31</UCPTdate>
                    <UCPTyearMask LonFormat="UCPTyearMask">DW_NUL</UCPTyearMask>
                    <UCPTmonthMask LonFormat="UCPTmonthMask">DW_NUL</UCPTmonthMask>
                    <UCPTdayMask LonFormat="UCPTdayMask">DW_NUL</UCPTdayMask>
                </EndDate>
                <UCPTschedDay LonFormat="UCPTschedDay">DM_DAY_4</UCPTschedDay>
                <UCPTschedMonth LonFormat="UCPTschedMonth">MN_JUL</UCPTschedMonth>
            </Schedule>
            <Schedule>
                <StartDate>
                    <UCPTdate>2008-03-12</UCPTdate>
                    <UCPTyearMask LonFormat="UCPTyearMask">DW_NUL</UCPTyearMask>
                    <UCPTmonthMask LonFormat="UCPTmonthMask">DW_NUL</UCPTmonthMask>
                    <UCPTdayMask LonFormat="UCPTdayMask">DW_NUL</UCPTdayMask>
                </StartDate>
                <EndDate>
                    <UCPTdate>2015-12-31</UCPTdate>
                    <UCPTyearMask LonFormat="UCPTyearMask">DW_NUL</UCPTyearMask>
                    <UCPTmonthMask LonFormat="UCPTmonthMask">DW_NUL</UCPTmonthMask>
                    <UCPTdayMask LonFormat="UCPTdayMask">DW_NUL</UCPTdayMask>
                </EndDate>
                <UCPTschedDay LonFormat="UCPTschedDay">DM_DAY_25</UCPTschedDay>
                <UCPTschedMonth LonFormat="UCPTschedMonth">MN_DEC</UCPTschedMonth>
            </Schedule>
            <UCPTmaxClient>1</UCPTmaxClient>
            <Client>
                <UCPTname>Net/LON/iLON App/Scheduler[0]</UCPTname>
                <UCPTservicePath/>
            </Client>
        </Exception>
    </Item>
```

10.2 Creating and Modifying the Calendar XML File

You can create and manage the #8000010128000000[4].UFPTcalendar.xml file with the *Set* SOAP function. The following section, *Calendar SOAP Interface*, describes how to use *Set* and the other SOAP functions provided for the Calendar application.

Alternatively, you can create and manage the #8000010128000000[4].UFPTcalendar.xml file manually with an XML editor and download it to the SmartServer via FTP. Echelon does not recommend this, as the SmartServer will require a reboot to read the configuration of the downloaded file. Additionally, the SmartServer 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 Calendar's configuration.

10.3 Calendar SOAP Interface

You can use the SOAP interface to perform the following functions on the Calendar:

Function	Description
List	Generate a list of the Calendars that you have added to the

	SmartServer.
Get	Retrieve the configuration of the Calendar that you have added to the SmartServer.
Set	Create a new Calendar, or overwrite the configuration of an existing Calendar.
Read	Retrieve information about exceptions for some period of time
Delete	Delete a Calendar.

Note: Section 21.1.3, *Creating a Scheduler and Calendar in Visual C# .NET*, includes a C# programming example demonstrating how to use the Calendar SOAP interface to create and read a data logger. Section 21.2.3, *Creating a Scheduler and Calendar in Visual Basic.NET*, includes a Visual Basic example demonstrating how to do this.

10.3.1 Using the List Function on a Calendar

You can use the *List* function to retrieve a list of the Calendars that you have added to the SmartServer. The List function takes an <iLonItem> element that includes an xSelect statement querying items of a UFPTcalendar_Cfg type as its input, as shown in the example below. The *List* function returns an <Item> element for each Calendar that you have added to the SmartServer. The next section describes the properties included in each of these elements.

You could use the list of <Item> elements returned by this function as input for the *Get* function. The *Get* function would then return the configuration of each Calendar included in the list.

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <xSelect>//Item[@xsi:type=" UFPTcalendar_Cfg"]</xSelect>
    </iLonItem>
</List>
```

Response

```
<ListResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
    <Item>
        <UCPTname>Net/LON/iLON App/Calendar</UCPTname>
        <UCPTannotation>#8000010128000000[4].UFPTcalendar;xsi:type=”LON_Fb_Cfg”</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
  </iLonItem>
 </ListResponse>
```

10.3.2 Using the Get Function a Calendar

You can use the *Get* function to retrieve the configuration of any Calendar that you have added to the SmartServer. You must reference the Calendar whose configuration is to be returned by its <UCPTname> in the input you supply to the function, as shown in the example below.

Request

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>Net/LON/iLON App/Calendar</UCPTname>
        </Item>
  </iLonItem>
</Get>
```

Response

```
<GetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="UFPTcalendar_Cfg" >
        <UCPTname>Net/LON/iLON App/Calendar</UCPTname>
        <UCPTannotation>#8000010128000000[4].UFPTcalendar;xsi:type="LON_Fb_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-05T16:09:15.700-08:00</UCPTlastUpdate>
        <UCPTuri>#8000010128000000[4].UFPTcalendar_Cfg.htm</UCPTuri>
        <ScheduleEffectivePeriod>
            <StartDate>2000-01-01</StartDate>
            <EndDate>2037-12-31</EndDate>
        </ScheduleEffectivePeriod>
        <Exception>
            <UCPTindex>4</UCPTindex>
            <UCPTexceptionName>Holiday</UCPTexceptionName>
            <UCPTaliasName>Holiday</UCPTaliasName>
            <UCPTtemporary>0</UCPTtemporary>
            <Schedule>
                <StartDate>
                    <UCPTdate>2008-03-12</UCPTdate>
                    <UCPTyearMask LonFormat="UCPTyearMask">DW_NUL</UCPTyearMask>
                    <UCPTmonthMask LonFormat="UCPTmonthMask">DW_NUL</UCPTmonthMask>
                    <UCPTdayMask LonFormat="UCPTdayMask">DW_NUL</UCPTdayMask>
                </StartDate>
                <EndDate>
                    <UCPTdate>2015-12-31</UCPTdate>
                    <UCPTyearMask LonFormat="UCPTyearMask">DW_NUL</UCPTyearMask>
                    <UCPTmonthMask LonFormat="UCPTmonthMask">DW_NUL</UCPTmonthMask>
                    <UCPTdayMask LonFormat="UCPTdayMask">DW_NUL</UCPTdayMask>
                </EndDate>
                <UCPTschedDay LonFormat="UCPTschedDay">DM_DAY_4</UCPTschedDay>
                <UCPTschedMonth LonFormat="UCPTschedMonth">MN_JUL</UCPTschedMonth>
            </Schedule>
            <Schedule>
                <StartDate>
                    <UCPTdate>2008-03-12</UCPTdate>
                    <UCPTyearMask LonFormat="UCPTyearMask">DW_NUL</UCPTyearMask>
                    <UCPTmonthMask LonFormat="UCPTmonthMask">DW_NUL</UCPTmonthMask>
                    <UCPTdayMask LonFormat="UCPTdayMask">DW_NUL</UCPTdayMask>
                </StartDate>
                <EndDate>
                    <UCPTdate>2015-12-31</UCPTdate>
                    <UCPTyearMask LonFormat="UCPTyearMask">DW_NUL</UCPTyearMask>
                    <UCPTmonthMask LonFormat="UCPTmonthMask">DW_NUL</UCPTmonthMask>
                    <UCPTdayMask LonFormat="UCPTdayMask">DW_NUL</UCPTdayMask>
                </EndDate>
                <UCPTschedDay LonFormat="UCPTschedDay">DM_DAY_25</UCPTschedDay>
                <UCPTschedMonth LonFormat="UCPTschedMonth">MN_DEC</UCPTschedMonth>
            </Schedule>
        </Exception>
    </Item>
  </iLonItem>
</GetResponse>
```

The function returns an <Item> element for each Calendar referenced in the input parameters supplied to the function. The properties included in each element are initially defined when the Calendar is created. You can write to these properties with the *Set* function. The following table lists and describes these properties.

Property	Description
<UCPTname>	The name of the Calendar in the following format: *<network/channel/device/functional block>*.
<UCPTannotation>	The program ID and functional profile template used by the

Property	Description
	Calendar. This property is always 8000010128000000[4].UFPTcalendar
<UCPThidden>	A flag indicating whether the Calendar functional block is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **0** – shown **1** – hidden
<UCPTitemStatus>	This property only appears if the data logger is not synchronized with an LNS network database or it has been deleted. In this case, it has the following values: **IS_NOTSYNCED** **IS_DELETED**
<UCPTlastUpdate>	A timestamp indicating the last time the configuration of the Calendar was updated. This timestamp uses the following format: YYYY-MM-DD**T**HH:MM:SS**Z**
<UCPTuri>	The name of the file containing the configuration web page for the Calendar on the SmartServer flash disk, absolute or relative to /web/user/echelon. This property is #8000010128000000[4].UFPTcalendar _Cfg.htm by default.
<ScheduleEffectivePeriod>	The <ScheduleEffectivePeriod> element contains two properties that define the dates that the Event Calendar applies to. The <StartDate> property defines the start date, and the <EndDate> property defines the end date. You must fill each property in using the following format: YYYY-MM-DD If the start date is undefined, it means any date up to and including the end date. If the end date is undefined, it means any date from the start date. If both are undefined, it means the Event Calendar is always active. The default <StartDate> is 2000-01-01. The default <EndDate> is 2037-12-31.

Property	Description
<Exception>	You can specify the dates that the Event Calendar applies to by creating exceptions. The exceptions that have been created for an Event Calendar are signified by a series of <Exception> elements. Each <Exception> element contains a group of <Schedule> child elements, each of which defines an exception date. The ability to create multiple <Schedule> elements allows you to create groups of exceptions that can be applied to a schedule together. For example, you may want to create a group of exceptions to apply to the first floor of a building, and another group of exceptions to apply to the second floor. In this case, you could specify two <Exception> elements, one for each floor. For a description of how to configure the properties you must define within each <Exception> element, see the next section, *Creating an Exception* **.**

10.3.2.1 Creating an Exception

The exception points for a Calendar are defined by a series of <Exception> elements.

```
<Exception>
    <UCPTindex>4</UCPTindex>
    <UCPTexceptionName>Holiday</UCPTexceptionName>
    <UCPTaliasName>Holiday</UCPTaliasName>
    <UCPTtemporary>0</UCPTtemporary>
    <Schedule></Schedule>
    <UCPTmaxClient>1</UCPTmaxClient>
    <Client>
        <UCPTname>Net/LON/iLON App/Scheduler[10]</UCPTname>
        <UCPTservicePath/>
    </Client>
</Exception>
```

The following table lists and describes the properties that must be defined within each <Exception> element.

Property	Description
<UCPTindex>	The index number assigned to the exception.
<UCPTexceptionName>	The name of the exception that you will use to reference it. This can be a maximum of 27 characters long.
<UCPTaliasName>	The name of the exception. There is no limit on the number and type of characters in this property.
<UCPTtemporary>	Either 0 or 1. If 0, this exception will be repeated annually. If 1, this will be a temporary exception. In this case, it will be removed from the Calendar, and any Schedulers referencing the exception, after the first time it is referenced.

Property	Description
<Schedule>	The <Schedule> element contains a series of child elements and properties that define the dates that the exception applies to. These are described in the next section, *Defining Exception Dates*.

10.3.2.2 Defining Exception Dates

The <Schedule> element contains a series of child elements and properties that define the dates that the Calendar is active.

```
<Schedule>
    <StartDate>
        <UCPTdate>2008-03-14</UCPTdate>
        <UCPTyearMask LonFormat="UCPTyearMask">DW_NUL</UCPTyearMask>
        <UCPTmonthMask LonFormat="UCPTmonthMask">DW_NUL</UCPTmonthMask>
        <UCPTdayMask LonFormat="UCPTdayMask">DW_NUL</UCPTdayMask>
    </StartDate>
    <EndDate>
        <UCPTdate>2037-03-31</UCPTdate>
        <UCPTyearMask LonFormat="UCPTyearMask">DW_WILDCARD</UCPTyearMask>
        <UCPTmonthMask LonFormat="UCPTmonthMask">DW_NUL</UCPTmonthMask>
        <UCPTdayMask LonFormat="UCPTdayMask">DW_NUL</UCPTdayMask>
    </EndDate>
    <UCPTschedDay LonFormat="UCPTschedDay">DM_EVERY_WEEKDAY</UCPTschedDay>
    <UCPTschedMonth LonFormat="UCPTschedMonth">MN_EVERY_MONTH</UCPTschedMonth>
</Schedule>
```

The following table lists and describes the elements and properties in a <Schedule> element:

<StartDate> <EndDate>	The <StartDate> and <EndDate> elements each contain the following properties that define the start and end dates for the exception: • <UCPTdate>. The date the exception starts or ends in the following format: YYYY-MM-DD. • <UCPTyearMask>. Indicates whether the exception applies to the years specified by the date range (DW_NUL), or to all years (DW_WILDCARD). • Setting the <StartDate> of <UCPTyearMask> to DW_ WILDCARD means that the starting year is 2000. • Setting the <EndDate> of <UCPTyearMask> to DW_ WILDCARD means that the ending year is 2037. • <UCPTmonthMask>. Indicates whether the exception applies to the pattern of months specified by the date range (DW_NUL), or to all months (DW_WILDCARD). • Setting the <StartDate> of <UCPTmonthMask> to DW_ WILDCARD means that the starting month is January. • Setting the <EndDate> of <UCPTmonthMask> to DW_ WILDCARD means that the ending month is December. • <UCPTdayMask>. Indicates whether the exception applies to the pattern of days marks specified by the date range (DW_NUL), or to all days (DW_WILDCARD). • Setting the <StartDate> of <UCPTdayMask> to DW_

	WILDCARD means that the starting date is the 1st. • Setting the <EndDate> of <UCPTdayMask> to DW_ WILDCARD means that the ending date is the 31st.
<UCPTschedDay>	Specifies the days of the month in which the exception will be valid during the interval specified by the <StartDate> and <EndDate> elements. For example, you could specify every third day during the interval, every fourth day, and so on. See *Daily Recursions* for a table listing the possible values for this property.
<UCPTschedMonth>	Specifies the months in which the exception will be valid during the interval specified by the <StartDate> and <EndDate> elements. For example, you could specify every other month during the interval, quarterly, and so on. See *Monthly Recursions* for a table listing the possible values for this property.

10.3.2.3 Daily Recursions

The following table lists and describes the values you can specify in the <UCPTschedDay> property of the <UCPTSchedule> element. The calendar will be active during the dates specified by this property.

Identifier	Description
DM_LAST_DAY_OF_MONTH	Last day of month
DM_LAST_SECOND_DAY	Second-to-last day of the month.
DM_LAST_THIRD_DAY	Third-to-last day of the month
NOTE: There are many other identifiers that use the DM_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 identifier DM_LAST_20_DAY to have the exception occur on the 20th to last day of each month the exception applies to.	
DM_LAST_30_DAY	30th to last day of the month
DM_FIRST_SUN	First Sunday of each month
DM_FIRST_MON	First Monday of each month
DM_SECOND_MON	Second Monday of each month
DM_THIRD_MON	Third Monday of each month
DM_FOURTH_MON	Fourth Monday of each month
DM_FIFTH_MON	Fifth Monday of each month
DM_FIRST_SAT	First Saturday of month
DM_SECOND_SUN	Second Sunday of each month

Identifier	Description
DM_SECOND_MON	Second Monday of each month
DM_SECOND_TUE	Second Tuesday of each month
DM_SECOND_WED	Second Wednesday of each month
DM_SECOND_THU	Second Thursday of each month
DM_SECOND_FRI	Second Friday of each month
DM_SECOND_SAT	Second Saturday of each month
DM_THIRD_SUN	Third Sunday of each month
DM_THIRD_MON	Third Monday of each month
DM_THIRD_TUE	Third Tuesday of each month
DM_THIRD_WED	Third Wednesday of each month
DM_THIRD_THU	Third Thursday of each month
DM_THIRD_FRI	Third Friday of each month
DM_THIRD_SAT	Third Saturday of each month
DM_FOURTH_SUN	Fourth Sunday of each month
DM_FOURTH_MON	Fourth Monday of each month
DM_FOURTH_TUE	Fourth Tuesday of each month
DM_FOURTH_WED	Fourth Wednesday of each month
DM_FOURTH_THU	Fourth Thursday of each month
DM_FOURTH_FRI	Fourth Friday of each month
DM_FOURTH_SAT	Fourth Saturday of each month
DM _FIFTH_SUN	Fifth Sunday of each month
DM_FIFTH_MON	Fifth Monday of each month
DM_FIFTH_TUE	Fifth Tuesday of each month
DM_FIFTH_WED	Fifth Wednesday of each month
DM_FIFTH_THU	Fifth Thursday of each month

Identifier	Description
DM_FIFTH_FRI	Fifth Friday of each month
DM_FIFTH_SAT	Fifth Saturday of each month
DM_LAST_SUN	Last Sunday of each month
DM_LAST_MON	Last Monday of each month
DM_LAST_TUE	Last Tuesday of each month
DM_LAST_WED	Last Wednesday of each month
DM_LAST_THU	Last Thursday of each month
DM_LAST_FRI	Last Friday of each month
DM_LAST_SAT	Last Saturday of each month
DM_EVERY_SUN	Every Sunday of the date interval.
DM_EVERY_MON	Every Monday of the date interval.
DM_EVERY_TUE	Every Tuesday of the date interval.
DM_EVERY_WED	Every Wednesday of the date interval.
DM_EVERY_THU	Every Thursday of the date interval.
DM_EVERY_FRI	Every Friday of the date interval.
DM_EVERY_SAT	Every Saturday of date interval
DM_EVERY_SECOND_DAY	Every second day of date interval
DM_EVERY_THIRD_DAY	Every third day of date interval
DM_EVERY_FOURTH_DAY	Every fourth day of the date interval
DM_EVERY_FIFTH_DAY	Every fifth day of the date interval
DM_EVERY_SIXTH_DAY	Every sixth day of the date interval
DM_NUL	Value not available. If this is chosen, the Calendar will use every day.

10.3.2.4 Monthly Recursions

The following table lists and describes the values you can specify in the <UCPTschedMonth> property of the <UCPTSchedule> element. The calendar will be active during the months specified by this property.

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 Calendar is active.
MN_EVERY_2_MONTH	Every other month during the interval the Calendar is active.
MN_QUARTERLY	Every third month during the interval the Calendar is active.
MN_EVERY_4_MONTH	Every fourth month during the interval the Calendar is active.
MN_EVERY_5_MONTH	Every fifth month during the interval the Calendar is active.
MN_EVERY_6_MONTH	Every sixth month during the interval the Calendar is active.
MN_EVERY_7_MONTH	Every seventh month during the interval the Calendar is active.
MN_EVERY_8_MONTH	Every eighth month during the interval the Calendar is active.
MN_EVERY_9_MONTH	Every ninth month during the interval the Calendar is active.
MN_EVERY_10_MONTH	Every tenth month during the interval the Calendar is

Identifier	Description
	active.
MN_EVERY_11_MONTH	Every eleventh month during the interval the Calendar is active.
MN_NUL	Value not available. If this is chosen, the Calendar will use every month.

10.3.2.5 Exception Examples

The following section presents a series of examples that demonstrate how to create exceptions that use wild cards and recursions.

Consider a case where you need to create an exception to apply to a specific range of dates, year after year. You could set the <UCPTyearMask> property in both the <StartDate> and <EndDate> elements to DW_WILDCARD. This applies the exception to all years, not just the ones specified by the <UCPTDate> element. The other properties are set to DW_NUL, so the exception applies to the days and months specified by the start and stop dates. As a result, the exception shown below applies to April 5th through 7th, every year.

```
<StartDate>
    <UCPTdate>2008-04-05</UCPTdate>
    <UCPTyearMask>DW_WILDCARD</UCPTyearMask>
    <UCPTmonthMask>DW_NUL</UCPTmonthMask>
    <UCPTdayMask>DW_NUL</UCPTdayMask>
</StartDate>
<EndDate>
    <UCPTdate>2008-04-07</UCPTdate>
    <UCPTyearMask>DW_WILDCARD</UCPTyearMask>
    <UCPTmonthMask>DW_NUL</UCPTmonthMask>
    <UCPTdayMask>DW_NUL</UCPTdayMask>
 </EndDate>
<UCPTschedDay LonFormat="UCPTschedDay">DM_NUL</UCPTschedDay>
<UCPTschedMonth LonFormat="UCPTschedMonth">MN_NUL</UCPTschedMonth>
```

Consider a case where you need to create an exception to apply to the first ten days of every month, year after year. You could do so by setting the <UCPTyearMask> and <UCPTmonthMask> properties to DW_WILDCARD, so the years and months specified in the start and stop dates are ignored. The <UCPTdayMask> property is set to DW_NUL, so the days specified (1 through 10) are used. As a result, the exception shown below applies to April 1st through April 10th, every year.

```
<StartDate>
    <UCPTdate>2008-04-01</UCPTdate>
    <UCPTyearMask>DW_WILDCARD</UCPTyearMask>
    <UCPTmonthMask>DW_WILDCARD</UCPTmonthMask>
    <UCPTdayMask>DW_NUL</UCPTdayMask>
</StartDate>
<EndDate>
    <UCPTdate>2008-04-10</UCPTdate>
    <UCPTyearMask>DW_WILDCARD</UCPTyearMask>
    <UCPTmonthMask>DW_WILDCARD</UCPTmonthMask>
     <UCPTdayMask>DW_NUL</UCPTdayMask>
 </EndDate>
<UCPTschedDay LonFormat="UCPTschedDay">DM_NUL</UCPTschedDay>
<UCPTschedMonth LonFormat="UCPTschedMonth">MN_NUL</UCPTschedMonth>
```

Now consider a case where you need to create an exception to apply to every weekday of every month, year after year. You could do so by setting the <UCPTyearMask>, <UCPTmonthMask>, and <UCPTschedDay> to DW_WILDCARD, the <UCPTschedDay> property to DM_EVERY_WEEKDAY, and the <UCPTschedMonth> property to MN_NUL.

```
<StartDate>
    <UCPTdate>2008-04-01</UCPTdate>
```

```
    <UCPTyearMask>DW_WILDCARD</UCPTyearMask>
    <UCPTmonthMask>DW_WILDCARD</UCPTmonthMask>
    <UCPTdayMask>DW_WILDCARD </UCPTdayMask>
</StartDate>
<EndDate>
    <UCPTdate>2008-05-31</UCPTdate>
    <UCPTyearMask>DW_WILDCARD</UCPTyearMask>
    <UCPTmonthMask>DW_WILDCARD</UCPTmonthMask>
    <UCPTdayMask> DW_WILDCARD</UCPTdayMask>
 </EndDate>
<UCPTschedDay LonFormat="UCPTschedDay">DM_FOURTH_SUN </UCPTschedDay>
<UCPTschedMonth LonFormat="UCPTschedMonth">MN_NUL</UCPTschedMonth>
```

Now consider a case where you need to create an exception to apply to every fourth Sunday of every month, for the next 10 years. You could do so by setting the <UCPTyearMask> to DW_NUL, the <UCPTmonthMask> to DW_WILDCARD and the <UCPTdayMask> to DW_NUL. You would also set the <UCPTschedDay> property to DM_FOURTH_SUN, and the <UCPTschedMonth> property to MN_NUL.

```
<StartDate>
    <UCPTdate>2008-04-01</UCPTdate>
    <UCPTyearMask> DW_NUL</UCPTyearMask>
    <UCPTmonthMask>DW_WILDCARD</UCPTmonthMask>
    <UCPTdayMask>DW_NUL</UCPTdayMask>
</StartDate>
<EndDate>
    <UCPTdate>2018-04-01</UCPTdate>
    <UCPTyearMask> DW_NUL</UCPTyearMask>
    <UCPTmonthMask>DW_WILDCARD</UCPTmonthMask>
    <UCPTdayMask>DW_NUL</UCPTdayMask>
 </EndDate>
<UCPTschedDay LonFormat="UCPTschedDay">DM_FOURTH_SUN </UCPTschedDay>
<UCPTschedMonth LonFormat="UCPTschedMonth">MN_NUL</UCPTschedMonth>
```

10.3.3 Using the Set Function on a Calendar

You can use the *Set* function to create new Calendars, or to overwrite the configuration of existing Calendars. The Calendars to be created or written to are signified by a list of <Item> elements in the input you supply to the function. The properties you must define within each <Item> element are the same, whether you are creating a new Calendar or modifying an existing Calendar. The previous section, *Using the Get Function on a Calendar*, describes these properties.

Note: If you specify a Calendar with the <UCPTname> element, the *Set* function deletes the specified Calendar before the specified parameters are set. If the <UCPTname> element is not specified, a new Calendar is created.

The first invocation of the *Set* function will generate the #8000010128000000[4].UFPTcalendar.xml file in the root/config/network/<network>/<channel>/iLONApp ||<device> directory of the SmartServer, if the file does not already exist.

When modifying an existing Calendar, any optional properties omitted from the input will be erased. Old values will not be preserved, so you should fill in every property when writing to a Calendar, even if you are not changing all of the values.

When creating or modifying a Calendar with this function, you may want to use output from the *Get* function as the basis for your input. 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 following example creates a Calendar with one exception that recurs every weekday day of every month, year after year.

Request

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTcalendar_Cfg">
```

```
        <UCPTname>Net/LON/iLON App/Calendar</UCPTname>
        <UCPTdescription>enter an optional description</UCPTdescription>
        <ScheduleEffectivePeriod xsi:type="UFPTscheduler_CfgEffectivePeriod">
            <StartDate>2000-01-01</StartDate>
            <EndDate>2037-12-31</EndDate>
        </ScheduleEffectivePeriod>
        <Exception>
            <UCPTindex>3</UCPTindex>
            <UCPTexceptionName>RecurringWeekendException</UCPTexceptionName>
            <UCPTaliasName>Recurring Weekend Exception</UCPTaliasName>
            <UCPTtemporary>0</UCPTtemporary>
            <Schedule>
                <StartDate xsi:type="UFPTcalendar_CfgESDate">
                    <UCPTdate>2008-03-06</UCPTdate>
                    <UCPTyearMask xsi:type="string" LonFormat="UCPTyearMask">DW_NUL</UCPTyearMask>
                    <UCPTmonthMask xsi:type="string" LonFormat="UCPTmonthMask">DW_NUL</UCPTmonthMask>
                    <UCPTdayMask xsi:type="string" LonFormat="UCPTdayMask">DW_NUL</UCPTdayMask>
                </StartDate>
                <EndDate xsi:type="UFPTcalendar_CfgESDate">
                    <UCPTdate>2037-12-31</UCPTdate>
                    <UCPTyearMask xsi:type="string" LonFormat="UCPTyearMask">DW_WILDCARD
                    </UCPTyearMask>
                    <UCPTmonthMask xsi:type="string" LonFormat="UCPTmonthMask">DW_WILDCARD
                    </UCPTmonthMask>
                    <UCPTdayMask xsi:type="string" LonFormat="UCPTdayMask">DW_WILDCARD</UCPTdayMask>
                </EndDate>
                <UCPTschedDay xsi:type="string" LonFormat="UCPTschedDay">DM_EVERY_WEEKEND_DAY
                </UCPTschedDay>
                <UCPTschedMonth xsi:type="string" LonFormat="UCPTschedMonth">MN_NUL</UCPTschedMonth>
            </Schedule>
        </Exception>
    </Item>
  </iLonItem>
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/"><iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/LON/iLON App/Calendar__TEMP_OBJECT</UCPTname>
    </Item>
</iLonItem>
</SetResponse>
```

10.3.4 Using the Read Function on a Calendar

You can use the *Read* function to retrieve the events scheduled in the exceptions on a Calendar on the SmartServer. Optionally, you can filter events using an xSelect statement and specifying one or more Calendar items. If you do not filter the results with an xSelect statement, the *Read* function returns the first 50 events in the selected calendar starting from January 1st, 2000.

You can use the following filters in an xSelect statement when using the *Read* function on a Calendar:

UCPTlastUpdate	When an event occurs. You can compare the specified start and stop times using equal, less (or equal), great (or equal). UCPTstop is set to time of last shown event. You need to check whether UCPTstop differs from UCPTlastUpdate to make the next request start from the UCPTstop time.
UCPTexeptionName	The name of the exception in which the events are scheduled.

position()	Position of the events in previously selected list. You can use this filter to limit the number of events in a result. You can compare the number of events to be returned using equal, less (or equal), or great (or equal). You can request a maximum of 500 events, however, more than 500 events may be returned. **Note**: The last instance of an exception before the specified start date is always returned (if it exists), and it is not counted by the position() constraint.

The following code demonstrates how to use the *Read* function to return a list of up to the first 100 events on a Calendar on the SmartServer:

Request

```
<Read xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="UFPTcalendar_Data"]
                   [UCPTlastUpdate&gt;="2008-03-07T00:00:00.000"
                   and UCPTlastUpdate&lt;="2037-12-31T00:00:00.000"]
                   [position()&lt;100]
    </xSelect>
    <Item>
        <UCPTname>Net/LON/iLON App/Calendar</UCPTname>
    </Item>
  </iLonItem>
</Read>
```

Response

```
<ReadResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="UFPTcalendar_Data" >
        <UCPTname>Net/LON/iLON App/Calendar</UCPTname>
        <UCPTstart>2008-03-07T00:00:00.000-08:00</UCPTstart>
        <UCPTstop>2008-09-01T00:00:00.000-07:00</UCPTstop>
        <UCPTexceptionName>HolidayScheduler</UCPTexceptionName>
        <DateEvent LonFormat="UNVT_date_event">80 80 HolidayScheduler</DateEvent>
        <DateEvent LonFormat="UNVT_date_event">178 178 HolidayScheduler</DateEvent>
    </Item>
    <Item xsi:type="UFPTcalendar_Data" >
        <UCPTname>Net/LON/iLON App/Calendar</UCPTname>
        <UCPTstart>2008-03-07T00:00:00.000-08:00</UCPTstart>
        <UCPTstop>2008-04-27T00:00:00.000-07:00</UCPTstop>
        <UCPTexceptionName>Inventory</UCPTexceptionName>
        <DateEvent LonFormat="UNVT_date_event">23 23 Inventory</DateEvent>
        <DateEvent LonFormat="UNVT_date_event">51 51 Inventory</DateEvent>
    </Item>
  </iLonItem>
</ReadResponse>
```

The *Read* function returns the following properties for each event from the specified Calendar:

<UCPTname>	The name of the calendar from which events were read in the following format: <*network*/*channel*/*device*/*functional block*>.
<UCPTstart> <UCPTstop>	Timestamps indicating the times of the first and last dates of the events returned by the *Read* function.
<UCPTexceptionName>	The name of the exception from which events were read.
<DateEvent>	A set of structures containing signed longs indicating the number of days from <UCPTstart> to the first and last

	events within the timeframe from <UCPTstart> to <UCPTstop>. The first <DateEvent> element specifies the number of days from <UCPTstart> to the first event in the specified timeframe. The first signed long in the structure indicates the number of days until the event is valid, and the second one indicates the days until the event is invalid. The second <DateEvent> element specifies the number of days from <UCPTstart> to the last event in the specified timeframe.

10.3.5 Using the Delete Function on a Calendar

You can use the *Delete* function to delete a Calendar. To delete a Calendar, you provide an <Item> element with a UFPTcalendar_Cfg type that includes the <UCPTname> property of the calendar to be deleted. The following code sample demonstrates how to use the *Delete* function to delete a Calendar:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTcalendar_Cfg">
        <UCPTname>Net/LON/iLON App/Calendar1</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/LON/iLON App/Calendar1</UCPTname>
    </Item>
  </iLonItem>
</DeleteResponse>
```

11 Real-Time Clock

You can use the real-time clock on the SmartServer to schedule events to start or stop based on the calculated sundown or sunrise, or a configured amount of time before or after the sundown or sunrise. The real-time clock includes an astronomical position sensor that calculates the position of the sun based on the time-of-day stored on the SmartServer and the location (geographic coordinates) of the SmartServer, which you specify. Based on the calculated position of the sun, the real-time clock can determine the sunrise and sundown times and pass this information to the Schedulers on the SmartServer.

11.1 Overview of the Real-Time Clock XML File

The #8000010128000000[4].UFPTrealTimeClock.xml file stores the configuration of the real-time clock on the SmartServer. You can create multiple real-time clocks; however, the SmartServer supports only 1 active real-time clock at a time. The following represents a sample #8000010128000000[4].UFPTrealTimeClock.xml file for a SmartServer:

```
<Item xsi:type="UFPTrealTimeClock_Cfg" >
    <UCPTname>Net/LON/iLON App/Real Time Clock</UCPTname>
    <UCPTannotation>#8000010128000000[4].UFPTrealTimeClock;xsi:type="LON_Fb_Cfg"</UCPTannotation>
    <UCPThidden>0</UCPThidden>
    <UCPTlastUpdate>2008-03-07T14:21:09.390-08:00</UCPTlastUpdate>
    <UCPTuri>#8000010128000000[4].UFPTrealTimeClock_Cfg.htm</UCPTuri>
    <DataPoint dpType="nvoTimeDate" discrim="dir_out" >
        <UCPTname>Net/LON/iLON App/Real Time Clock/nvoRtTimeDate</UCPTname>
    </DataPoint>
    <DataPoint dpType="nviTimeSet" discrim="dir_in" >
        <UCPTname>Net/LON/iLON App/Node Object/nviTimeSet</UCPTname>
    </DataPoint>
    <DataPoint dpType="nciMasterSlave" discrim="dir_in" >
        <UCPTname>Net/LON/iLON App/Real Time Clock/nciRtMasterSlave</UCPTname>
    </DataPoint>
    <DataPoint dpType="nciUpdateRate" discrim="dir_in" >
        <UCPTname>Net/LON/iLON App/Real Time Clock/nciRtUpdateRate</UCPTname>
    </DataPoint>
    <DataPoint dpType="nvoSummerTime" discrim="dir_out" >
        <UCPTname>Net/LON/iLON App/Real Time Clock/nvoRtSummerTime</UCPTname>
    </DataPoint>
    <DataPoint dpType="nvoWinterTime" discrim="dir_out" >
        <UCPTname>Net/LON/iLON App/Real Time Clock/nvoRtWinterTime</UCPTname>
    </DataPoint>
    <DataPoint dpType="nviTimeZone" discrim="dir_in" >
        <UCPTname>Net/LON/iLON App/Real Time Clock/nviRtTimeZone</UCPTname>
    </DataPoint>
    <DataPoint dpType="Elevation" discrim="dir_out" >
        <UCPTname>Net/LON/iLON App/Real Time Clock/nvoElevation</UCPTname>
    </DataPoint>
    <DataPoint dpType="Azimuth" discrim="dir_out" >
        <UCPTname>Net/LON/iLON App/Real Time Clock/nvoAzimuth</UCPTname>
    </DataPoint>
    <DataPoint dpType="Sunrise" discrim="dir_out" >
        <UCPTname>Net/LON/iLON App/Real Time Clock/nvoSunrise</UCPTname>
    </DataPoint>
    <DataPoint dpType="Sunset" discrim="dir_out" >
        <UCPTname>Net/LON/iLON App/Real Time Clock/nvoSunset</UCPTname>
    </DataPoint>
        <UCPTdegLongitude>-121.913</UCPTdegLongitude>
        <UCPTdegLatitude>37.3182</UCPTdegLatitude>
</Item>
```

11.2 Creating and Modifying the Real-Time Clock XML File

You can create and manage the #8000010128000000[4].UFPTrealTimeClock.xml file with the *Set* SOAP function. The following section, *Real-Time Clock SOAP Interface*, describes how to use *Set* and the other SOAP functions provided for the real-time clock application.

Alternatively, you can create and manage the #8000010128000000[4].UFPTrealTimeClock.xml file manually with an XML editor and download it to the SmartServer via FTP. Echelon does not recommend this, as the SmartServer will require a reboot to read the configuration of the downloaded file. Additionally, the SmartServer 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 real-time clock's configuration.

11.3 Real-Time Clock SOAP Interface

You can use the SOAP interface to perform the following functions on the real-time clock:

Function	Description
List	Generate a list of the real-time clocks on the SmartServer.
Get	Retrieve the configuration of the real-time clocks on the SmartServer.
Set	Create a new real-time clock, or overwrite the configuration of an existing real-time clock.
Delete	Delete a real-time clock.

11.3.1 Using the List Function on a Real-Time Clock

You can use the *List* function to retrieve a list of the real-time clocks that you have added to the SmartServer. The *List* function takes an <iLonItem> element that includes an xSelect statement querying items with a UFPTrealTimeClock_Cfg type as its input, as shown in the example below. The *List* function returns an <Item> element for each real-time clock that you have added to the SmartServer. The next section describes the properties included in each of these elements.

You could use the list of <Item> elements returned by this function as input for the *Get* function. The *Get* function would then return the configuration of each real-time clock included in the list.

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <xSelect>//Item[@xsi:type=" UFPTrealTimeClock_Cfg"]</xSelect>
    </iLonItem>
</List>
```

Response

```
<ListResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
    <Item>
        <UCPTname>Net/LON/iLON App/RealTimeClock</UCPTname>
        <UCPTannotation>#8000010128000000[4].UFPTrealTimeClock;xsi:type="LON_Fb_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
  </iLonItem>
 </ListResponse>
```

11.3.2 Using the Get Function on a Real-Time Clock

You can use the *Get* function to retrieve the configuration of any real-time clock that you have added to the SmartServer. You must reference the real-time clock whose configuration is to be returned by its <UCPTname> in the input you supply to the function, as shown in the example below.

Request

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>Net/LON/iLON App/Real Time Clock</UCPTname>
    </Item>
  </iLonItem>
</Get>
```

Response

```
<GetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="UFPTrealTimeClock_Cfg" >
        <UCPTname>Net/LON/iLON App/Real Time Clock</UCPTname>
        <UCPTannotation>#8000010128000000[4].UFPTrealTimeClock;xsi:type="LON_Fb_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-07T14:21:09.390-08:00</UCPTlastUpdate>
        <UCPTuri>#8000010128000000[4].UFPTrealTimeClock_Cfg.htm</UCPTuri>
        <DataPoint dpType="nvoTimeDate" discrim="dir_out" >
            <UCPTname>Net/LON/iLON App/Real Time Clock/nvoRtTimeDate</UCPTname>
        </DataPoint>
        <DataPoint dpType="nviTimeSet" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Node Object/nvoIpAddress#2</UCPTname>
        </DataPoint>
        <DataPoint dpType="nciMasterSlave" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Real Time Clock/nciRtMasterSlave</UCPTname>
        </DataPoint>
        <DataPoint dpType="nciUpdateRate" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Real Time Clock/nciRtUpdateRate</UCPTname>
        </DataPoint>
        <DataPoint dpType="nvoSummerTime" discrim="dir_out" >
            <UCPTname>Net/LON/iLON App/Real Time Clock/nvoRtSummerTime</UCPTname>
        </DataPoint>
        <DataPoint dpType="nvoWinterTime" discrim="dir_out" >
            <UCPTname>Net/LON/iLON App/Real Time Clock/nvoRtWinterTime</UCPTname>
        </DataPoint>
        <DataPoint dpType="nviTimeZone" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/Real Time Clock/nviRtTimeZone</UCPTname>
        </DataPoint>
        <DataPoint dpType="Elevation" discrim="dir_out" >
            <UCPTname>Net/LON/iLON App/Real Time Clock/nvoElevation</UCPTname>
        </DataPoint>
        <DataPoint dpType="Azimuth" discrim="dir_out" >
            <UCPTname>Net/LON/iLON App/Real Time Clock/nvoAzimuth</UCPTname>
        </DataPoint>
        <DataPoint dpType="Sunrise" discrim="dir_out" >
            <UCPTname>Net/LON/iLON App/Real Time Clock/nvoSunrise</UCPTname>
        </DataPoint>
        <DataPoint dpType="Sunset" discrim="dir_out" >
            <UCPTname>Net/LON/iLON App/Real Time Clock/nvoSunset</UCPTname>
        </DataPoint>
        <UCPTdegLongitude>-121.913</UCPTdegLongitude>
        <UCPTdegLatitude>37.3182</UCPTdegLatitude>
    </Item>
  </iLonItem>
</GetResponse>
```

The function returns an <Item> element for each real-time clock referenced in the input parameters supplied to the function. The properties included in each element are initially defined when the real-time clock is created. You can write to these properties with the *Set* function. The following table describes these properties.

Property	Description
<UCPTname>	The name of the real-time clock in the following format: <*network*/*channel*/*device*/*functional block*>.

Property	Description
<UCPTannotation>	The program ID and functional profile template used by the real-time clock. This property is always 8000010128000000[4].UFPTrealTimeClock
<UCPThidden>	A flag indicating whether the real-time clock functional block is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **0** – shown **1** – hidden
<UCPTitemStatus>	This property only appears if the data logger is not synchronized with an LNS network database or it has been deleted. In this case, it has the following values: **IS_NOTSYNCED** **IS_DELETED**
<UCPTlastUpdate>	A timestamp indicating the last time the configuration of the real-time clock was updated. This timestamp uses the following format: YYYY-MM-DD**T**HH:MM:SS**Z**
<UCPTuri>	The name of the file containing the configuration web page for the Real-Time Clock on the SmartServer flash disk, absolute or relative to /web/user/echelon. This property is #8000010128000000[4].UFPTrealTimeClock _Cfg.htm by default.
<UCPTdegLongitude>	The north-south location of the SmartServer relative to the equator as a decimal fraction. If the SmartServer is located south of the equator, this is a negative value between 0 and –90. If it is located north of the equator, it is a positive value between 0 and 90.
<UCPTdegLatitude>	The east-west location of the SmartServer relative to the Prime Meridian as a decimal fraction. If the SmartServer is located west of the Prime Meridian, this is a negative value between 0 and –180. If it is located is located east of the Prime Meridian, it is a positive value between 0 and 180.

11.3.3 Using the Set Function on a Real-Time Clock

You can use the *Set* function to create new real-time clocks, or to overwrite the configuration of existing real-time clocks. The real-time clocks to be created or written to are signified by a list of <Item> elements in the input you supply to the function. The properties you must define within each <Item> element are the same, whether you are creating a new real-time clock or modifying an existing real-time clock. The previous section, *Using the Get Function on a Real-Time Clock*, describes these properties.

Note: If you specify a real-time clock with the <UCPTname> element, the *Set* function deletes the specified real-time clock before the specified parameters are set. If the <UCPTname> element is not specified, a new real-time clock is created.

The first invocation of the *Set* function will generate the #8000010128000000[4].UFPTrealTimeClock.xml file in the root/config/network/<network>/<channel>/iLONApp ||<device> directory of the SmartServer, if the file does not already exist.

When modifying an existing real-time clock, any optional properties omitted from the input will be erased. Old values will not be preserved, so you should fill in every property when writing to a real-time clock, even if you are not changing all of the values.

When creating or modifying a real-time clock with this function, you may want to use output from the *Get* function as the basis for your input. 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.

Request

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTrealTimeClock_Cfg">
        <UCPTname>Net/LON/iLON App/Real Time Clock</UCPTname>
        <UCPTdescription>enter an optional description</UCPTdescription>
        <DataPoint dpType="Elevation" discrim="dir_out">
            <UCPTname>Net/LON/iLON App/Real Time Clock/nvoElevation</UCPTname>
        </DataPoint>
        <DataPoint dpType="Azimuth" discrim="dir_out">
            <UCPTname>Net/LON/iLON App/Real Time Clock/nvoAzimuth</UCPTname>
        </DataPoint>
        <DataPoint dpType="Sunrise" discrim="dir_out">
            <UCPTname>Net/LON/iLON App/Real Time Clock/nvoSunrise</UCPTname>
        </DataPoint>
        <DataPoint dpType="Sunset" discrim="dir_out">
            <UCPTname>Net/LON/iLON App/Real Time Clock/nvoSunset</UCPTname>
        </DataPoint>
        <UCPTdegLongitude>-121.913</UCPTdegLongitude>
        <UCPTdegLatitude>37.3182</UCPTdegLatitude>
    </Item>
  </iLonItem>
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/LON/iLON App/Real Time Clock</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

11.3.4 Using the Delete Function on a Real-Time Clock

You can use the *Delete* function to delete a Real-Time Clock. To delete a Real-Time Clock, you provide an <Item> element with a UFPTrealTimeClock_Cfg type that includes the <UCPTname> property of the real-Time clock to be deleted. The following code sample demonstrates how to use the *Delete* function to delete a Real-Time Clock:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTrealTimeClock_Cfg">
        <UCPTname>Net/LON/iLON App/Real Time Clock</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/LON/iLON App/Real Time Clock</UCPTname>
    </Item>
  </iLonItem>
</DeleteResponse>
```

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 accept as input, and the network variable type it will convert these data points to. The Type Translator Rule defines 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 SmartServer software includes nine pre-defined Type Translator Rules. Each one is described 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 when no offset or multipliers are required, or 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 SmartServer software, with the TypeTranslator_Rule SOAP functions. For more information on creating Type Translator Rules, or on modifying the Type Translator Rules provided with the SmartServer software, see Chapter 13, *Type Translator Rules*.

12.1 Overview of the Type Translator XML File

The #8000010128000000[4].UFPTtypeTranslator.xml file stores the configuration of all Type Translators you have added to the SmartServer. You can create new Schedulers using the *Set* function, or by manually editing the #8000010128000000[4].UFPTtypeTranslator.xml file, and rebooting the SmartServer.

You can create up to 40 Type Translators per SmartServer. You can add more than 40 Type Translators if you load the dynamic v40 XIF on your SmartServer and you operate your SmartServer in Standalone mode. Note that using the v40 XIF with the SmartServer operating in LNS mode (**LNS Auto** or **LNS Manual**) is not supported.

The following represents a sample #8000010128000000[4].UFPTtypeTranslator.xml file for a SmartServer with one Type Translator that translates a **SNVT_temp** data point to a **SNVT_temp_p** data point.

```
<Item xsi:type="UFPTtypeTranslator_Cfg" >
    <UCPTname>Net/LON/iLON App/Type Translator[0]</UCPTname>
    <UCPTannotation>8000010128000000[4].UFPTtypeTranslator</UCPTannotation>
    <UCPThidden>0</UCPThidden>
    <UCPTlastUpdate>2008-03-07T17:19:37.790-08:00</UCPTlastUpdate>
    <UCPTuri>#8000010128000000[4].UFPTtypeTranslator_Cfg.htm</UCPTuri>
    <DataPoint xsi:type="UFPTtypeTranslator_DpRef" dpType="Input" discrim="dir_in" >
        <UCPTname>Net/LON/iLON App/VirtFb/temp_thermostat</UCPTname>
        <UCPTformatDescription>#0000000000000000[0].SNVT_temp</UCPTformatDescription>
        <UCPTnickName>temp_thermostat</UCPTnickName>
    </DataPoint>
    <DataPoint xsi:type="UFPTtypeTranslator_DpRef" dpType="Output" discrim="dir_out" >
        <UCPTname>Net/LON/iLON App/VirtFb/temp_chiller</UCPTname>
        <UCPTformatDescription>#0000000000000000[0].SNVT_temp_p</UCPTformatDescription>
        <UCPTnickName>temp_chiller</UCPTnickName>
    </DataPoint>
    <SCPTdelayTime>0</SCPTdelayTime>
</Item>
```

12.2 Creating and Modifying the Type Translator XML File

You can create and manage the #8000010128000000[4].UFPTtypeTranslator.xml file with the *Set* SOAP function. The following section, *Type Translator SOAP Interface*, describes how to use *Set* and the other SOAP functions provided for the Type Translator application.

Alternatively, you can create and manage the #8000010128000000[4].UFPTtypeTranslator.xml file manually with an XML editor and download it to the SmartServer via FTP. Echelon does not recommend this, as the SmartServer will require a reboot to read the configuration of the downloaded file. Additionally, the SmartServer 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's configuration.

12.3 Type Translator SOAP Interface

You can use the SOAP interface to perform the following functions on a type translator:

Function	Description
List	Generate a list of the type translators on the SmartServer.
Get	Retrieve the configuration of the type translators on the SmartServer.
Set	Create a new type translator, or overwrite the configuration of an existing type translators.
Delete	Delete a type translators.

12.3.1 Using the List Function on a Type Translator

You can use the *List* function to retrieve a list of the type translators that you have added to the SmartServer. The List function takes an <iLonItem> element that includes an xSelect statement querying items of UFPTtypeTranslator_Cfg type as its input, as shown in the example below. The *List* function returns an <Item> element for each type translator that you have added to the SmartServer. The next section describes the properties included in each of these elements.

You could use the list of <Item> elements returned by this function as input for the *Get* function. The *Get* function would then return the configuration of each type translator included in the list.

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <xSelect>//Item[@xsi:type=" UFPTtypeTranslator_Cfg"]</xSelect>
    </iLonItem>
</List>
```

Response

```
<ListResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
    <Item>
        <UCPTname>Net/LON/iLON App/Type Translator[0]</UCPTname>
        <UCPTannotation>#8000010128000000[4].UFPTtypeTranslator;xsi:type=”LON_Fb_Cfg”
        </UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
```

```
  </iLonItem>
 </ListResponse>
```

12.3.2 Using the Get Function on a Type Translator

You can use the *Get* function to retrieve the configuration of any type translator that you have added to the SmartServer. You must reference the type translator whose configuration is to be returned by its <UCPTname> in the input you supply to the function, as shown in the example below.

Request

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>Net/LON/iLON App/Type Translator[0]</UCPTname>
    </Item>
  </iLonItem>
</Get>
```

Response

```
<GetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="UFPTtypeTranslator_Cfg" >
        <UCPTname>Net/LON/iLON App/Type Translator[0]</UCPTname>
        <UCPTannotation>8000010128000000[4].UFPTtypeTranslator</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-07T17:19:37.790-08:00</UCPTlastUpdate>
        <UCPTuri>#8000010128000000[4].UFPTtypeTranslator_Cfg.htm</UCPTuri>
        <DataPoint xsi:type="UFPTtypeTranslator_DpRef" dpType="Input" discrim="dir_in" >
            <UCPTname>Net/LON/iLON App/VirtFb/temp_thermostat</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_temp</UCPTformatDescription>
            <UCPTnickName>temp_thermostat</UCPTnickName>
        </DataPoint>
        <DataPoint xsi:type="UFPTtypeTranslator_DpRef" dpType="Output" discrim="dir_out" >
            <UCPTname>Net/LON/iLON App/VirtFb/temp_chiller</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_temp_p</UCPTformatDescription>
            <UCPTnickName>temp_chiller</UCPTnickName>
        </DataPoint>
        <UCPTtranslatorRule>My Custom Rule: SNVT_temp to SNVT_temp_p</UCPTtranslatorRule>
        <SCPTdelayTime>0</SCPTdelayTime>
    </Item>
  </iLonItem>
</GetResponse>
```

The function returns an <Item> element for each type translator referenced in the input parameters supplied to the function. The properties included in each element are initially defined when the type translator is created. You can write to these properties with the *Set* function. The following table describes these properties.

Property	Description
<UCPTname>	The name of the real-time clock in the following format: <*network*/*channel*/*device*/*functional block*>.
<UCPTannotation>	The program ID and functional profile template used by the real-time clock. This property is always 8000010128000000[4].UFPTrealTimeClock
<UCPThidden>	A flag indicating whether the real-time clock functional block is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **0** – shown

Property	Description
	1 – hidden
<UCPTitemStatus>	This property only appears if the data logger is not synchronized with an LNS network database or it has been deleted. In this case, it has the following values: **IS_NOTSYNCED** **IS_DELETED**
<UCPTlastUpdate>	A timestamp indicating the last time the configuration of the real-time clock was updated. This timestamp uses the following format: YYYY-MM-DD**T**HH:MM:SS**Z**
<UCPTuri>	The name of the file containing the configuration web page for the Type Translator on the SmartServer flash disk, absolute or relative to /web/user/echelon. This property is #8000010128000000[4].UFPTtypeTranslator_Cfg.htm by default.
<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 has multiple input 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.
<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 accept 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. Chapter 12 describes the pre-defined type translator rules included with the SmartServer 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. 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

Property	Description
	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.
<DataPoint> Input Output	The input and output data points that the Type Translator will translate are signified by a list of <DataPoint> elements. Each <DataPoint> element contains the following three properties: • <UCPTName>. The name of the data point in the following format: <*network*/*channel*/*device*/*functional block*/*data point*>. • <UCPTformatDescription>. The data point's program ID; data type (SNVT, SCPT, UNVT, UCPT, or built-in data type); and format (e.g., SI metric or US customary if the type has multiple formats such as **SNVT_temp_p**). The format description is displayed in the following format: • #<manufacturer ID>**[**scope selector**].**<type name>[#format] • <UCPTnickname>. A user-defined name for the data point that is used to reference the data point. You can use this property to reference a data point in the <InputPath> and <OutputPath> properties in a type translation. By default, the nickname of the data point is the data point's name. • For example, a data point with a <UCPTname> of "Net/LON/iLON App/VirtFb/nvoLevDisc" has a default <UCPTnickname> of "nvoLevDisc". Chapter 12 describes the Type Translator Rules provided with the SmartServer software, and the format types that each rule requires for the input data points.

12.3.3 Using the Set Function on a Type Translator

You can use the *Set* 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 <Item> elements in the input you supply to the function. The properties you must define within each <Item> element are the same, whether you are creating a new type translator or modifying an existing type translator. The previous section, *Using the Get Function on a Type Translator*, describes these properties.

Note: If you specify a type translator with the <UCPTname> element, the *Set* function deletes the specified type translator before the specified parameters are set. If the <UCPTname> element is not specified, a new type translator is created.

The first invocation of the *Set* function will generate the #8000010128000000[4].UFPTtypeTranslator.xml file in the root/config/network/<network>/<channel>/iLONApp ||<device> directory of the SmartServer, if the file does not already exist.

When modifying an existing type translator, any optional properties omitted from the input will be erased. Old values will not be preserved, so you should 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 this function, you may want to use output from the *Get* function as the basis for your input. 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 following uses the *Set* function to create a Type Translator that uses the Type Translator Rule "**SNVT_switch_TO_SNVT_lev_disc**" to translate a **SNVT_switch** data point (nviSwitch) to a **SNVT_lev_disc** data point (nvoLevDisc). Because the "**SNVT_switch_TO_SNVT_lev_disc**" rule is being used, nviSwitch must be a **SNVT_switch** data point and 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 SmartServer software are listed in Chapter 13.

Request

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTtypeTranslator_Cfg">
        <UCPTname>Net/LON/iLON App/Type Translator[1]</UCPTname>
        <UCPTdescription>enter an optional description</UCPTdescription>
        <DataPoint xsi:type="UFPTtypeTranslator_DpRef" dpType="Input">
            <UCPTname>Net/LON/iLON App/VirtFb/nviSwitch</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
            <UCPTnickName>Input0</UCPTnickName>
        </DataPoint>
        <DataPoint xsi:type="UFPTtypeTranslator_DpRef" dpType="Output">
            <UCPTname>Net/LON/iLON App/VirtFb/nvoLevDisc</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_lev_disc</UCPTformatDescription>
            <UCPTnickName>Output0</UCPTnickName>
        </DataPoint>
        <UCPTtranslatorRule>SNVT_switch_TO_SNVT_lev_disc</UCPTtranslatorRule>
        <SCPTdelayTime>0</SCPTdelayTime>
    </Item>
  </iLonItem>
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/LON/iLON App/Type Translator[1]</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

12.3.4 Pre-Defined Type Translator Rules

The following sections list the identifiers you can use to fill in the <UCPTtranslatorRule> property of a UFPTtypeTranslator_Cfg item when you create a new Type Translator. They also provide descriptions of the Type Translator Rules these identifiers reference, and 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 SmartServer.

12.3.4.1 16xSNVT_switch_TO_SNVT_state

You can use this Type Translator Rule to convert up to 16 **SNVT_switch** input data points to a single **SNVT_state** output data point. 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 16 **SNVT_switch** data points to be translated are defined by a list of <DataPoint> elements that have a "Dp Type" attribute of "Input". The input data points referenced by <UCPTname> must have a <UCPTformatDescription> property of **#0000000000000000[0]. SNVT_switch** data point and a <UCPTnickName> in the range of Input0 to Input15.
- The SNVT_state output data point is defined by a <DataPointFormat> element that have a "Dp Type" attribute of "Output". The output data point referenced by <UCPTname> must have a <UCPTformatDescription> property of **#0000000000000000[0]. SNVT_state** and a <UCPTnickName> of "Output0".

The value of the *state* field of each input data point will be read and stored in bitX of the output data point, where X represents the <UCPTnickName> of the input data point. For example, the *state* field of the **SNVT_switch** data point that has a <UCPTnickName> of Input0 would be stored in Bit0 of the output **SNVT_state** data point. Or, the *state* field of the **SNVT_switch** data point that has a <UCPTnickName> of Input8 would be stored in Bit8 of the output **SNVT_state** data point.

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

The following code demonstrates how to use the *Set* function to create a type translator that uses this <UCPTtranslatorRule>:

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTtypeTranslator_Cfg">
        <UCPTname>Net/LON/iLON App/Type Translator[0]</UCPTname>
        <DataPoint dpType="Input">
            <UCPTname>Net/LON/iLON App/VirtFb/nviSwitch_1</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
            <UCPTnickName>Input0</UCPTnickName>
        </DataPoint>
        <DataPoint dpType="Input">
            <UCPTname>Net/LON/iLON App/VirtFb/nviSwitch_2</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
            <UCPTnickName>Input1</UCPTnickName>
        </DataPoint>
        <DataPoint dpType="Output">
            <UCPTname>Net/LON/iLON App/VirtFb/nvoState</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_state</UCPTformatDescription>
            <UCPTnickName>Output0</UCPTnickName>
        </DataPoint>
        <UCPTtranslatorRule>16xSNVT_switch_TO_SNVT_state</UCPTtranslatorRule>
        <SCPTdelayTime>0</SCPTdelayTime>
    </Item>
  </iLonItem>
</Set>
```

12.3.4.2 SNVT_lev_disc_TO_SNVT_occupancy

You can 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**.

- The **SNVT_lev_disc** input data point to be translated is defined by a <DataPoint> element that have a "Dp Type" attribute of "Input". The input data point referenced by <UCPTname> must have a <UCPTformatDescription> of **#0000000000000000[0]. SNVT_lev_disc** and a <UCPTnickName> of Input0.
- The **SNVT_occupancy** output data point is defined by a <DataPointFormat> element that has a "Dp Type" attribute of "Output". The output data point referenced by <UCPTname> must have a <UCPTformatDescription> property of **#0000000000000000[0]. SNVT_occupancy** and a <UCPTnickName> property of Output0.

Each time a type translation is made, the **SNVT_occupancy** output data point is assigned a value based on the current enumeration stored in the **SNVT_lev_desc** input data point, as described in the following table:

SNVT_lev_desc (input point)	SNVT_occupancy (ouput point)
ST_NUL	OC_NUL
ST_OFF	OC_UNOCCUPIED
ST_ON	OC_OCCUPIED
ST_HIGH	OC_BYPASS
ST_LOW or ST_MED	OC_STANDBY

The following code demonstrates how to use the *Set* function to create a type translator that uses this <UCPTtranslatorRule>:

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTtypeTranslator_Cfg">
        <UCPTname>Net/LON/iLON App/Type Translator[0]</UCPTname>
        <DataPoint dpType="Input">
            <UCPTname>Net/LON/iLON App/VirtFb/nvilevDisc</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_lev_disc</UCPTformatDescription>
            <UCPTnickName>Input0</UCPTnickName>
        </DataPoint>
            <DataPoint dpType="Output">
            <UCPTname>Net/LON/iLON App/VirtFb/nvoOccupancy</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_occupancy</UCPTformatDescription>
            <UCPTnickName>Output0</UCPTnickName>
        </DataPoint>
        <UCPTtranslatorRule>SNVT_lev_disc_TO_SNVT_occupancy</UCPTtranslatorRule>
        <SCPTdelayTime>0</SCPTdelayTime>
    </Item>
  </iLonItem>
</Set>
```

12.3.4.3 SNVT_lev_disc_TO_SNVT_switch

You can 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**.

- The **SNVT_lev_disc** input data point to be translated is defined by a <DataPoint> element that have a “Dp Type” attribute of “Input”. The input data point referenced by <UCPTname> must have a <UCPTformatDescription> of **#0000000000000000[0]. SNVT_lev_disc** and a <UCPTnickName> of Input0.
- The **SNVT_switch** output data point is defined by a <DataPointFormat> element that has a “Dp Type” attribute of “Output”. The output data point referenced by <UCPTname> must have a <UCPTformatDescription> property of **#0000000000000000[0]. SNVT_switch** and a <UCPTnickName> property of Output0.

Each time a type translation is made, the **SNVT_switch** output data point is assigned a value and state based on the current enumeration stored in the **SNVT_lev_desc** input data point, as described in the following table:

SNVT_lev_desc (input point)	SNVT_switch (ouput point)
ST_NUL	OFF
ST_OFF	value: 0.0 state: 0 (OFF)
ST_ON	value: 100.0 state: 1 (ON)

ST_HIGH	value: 75.0 state: 1 (ON)
ST_MED	value: 50.0 state: 1 (ON)
ST_LOW	value: 25.0 state: 1 (ON)

The following code demonstrates how to use the *Set* function to create a type translator that uses this <UCPTtranslatorRule>:

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTtypeTranslator_Cfg">
        <UCPTname>Net/LON/iLON App/Type Translator[0]</UCPTname>
        <DataPoint dpType="Input">
            <UCPTname>Net/LON/iLON App/VirtFb/nvilevDisc</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_lev_disc</UCPTformatDescription>
            <UCPTnickName>Input0</UCPTnickName>
        </DataPoint>
            <DataPoint dpType="Output">
            <UCPTname>Net/LON/iLON App/VirtFb/nvoSwitch</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
            <UCPTnickName>Output0</UCPTnickName>
        </DataPoint>
        <UCPTtranslatorRule>SNVT_lev_disc_TO_SNVT_switch</UCPTtranslatorRule>
        <SCPTdelayTime>0</SCPTdelayTime>
    </Item>
  </iLonItem>
</Set>
```

12.3.4.4 SNVT_occupancy_TO_SNVT_setting

You can use this Type Translator Rule to translate an input data point of type **SNVT_setting** to an output data point of type **SNVT_occupancy**.

- The **SNVT_occupancy** input data point is defined by a <DataPointFormat> element that has a "Dp Type" attribute of "Input". The input data point referenced by <UCPTname> must have a <UCPTformatDescription> property of **#0000000000000000[0]. SNVT_occupancy** and a <UCPTnickName> property of Input0.
- The **SNVT_setting** output data point to be translated is defined by a <DataPoint> element that have a "Dp Type" attribute of "Output". The output data point referenced by <UCPTname> must have a <UCPTformatDescription> of **#0000000000000000[0]. SNVT_setting** and a <UCPTnickName> of Output0.

Each time a type translation is made, the function, rotation, and setting fields of the **SNVT_setting** ouput data point are assigned values based on the current enumeration stored in the **SNVT_occupancy** input data point, as described in the following table:

SNVT_occupancy (input point)	**SNVT_setting (ouput point)**
OC_NUL	function: SET_STATE (enumerated value is 5) setting: 0 rotation: 0
OC_UNOCCUPIED	function: SET_STATE setting: 60 rotation: -80.01
OC_OCCUPIED	function: SET_STATE setting: 100 rotation: 80.24

OC_BYPASS	function: SET_STATE setting: 100 rotation: 80.24
OC_STANDBY	function: SET_STATE setting: 60.2 rotation: -40

The following code demonstrates how to use the *Set* function to create a type translator that uses this <UCPTtranslatorRule>:

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTtypeTranslator_Cfg">
        <UCPTname>Net/LON/iLON App/Type Translator[5]</UCPTname>
        <DataPoint dpType="Input">
            <UCPTname>Net/LON/iLON App/VirtFb/nviOccupancy_1</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_occupancy</UCPTformatDescription>
            <UCPTnickName>Input0</UCPTnickName>
        </DataPoint>
            <DataPoint dpType="Output">
            <UCPTname>Net/LON/iLON App/VirtFb/nvoSetting_1</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_setting</UCPTformatDescription>
            <UCPTnickName>Output0</UCPTnickName>
        </DataPoint>
        <UCPTtranslatorRule>SNVT_occupancy_TO_SNVT_setting</UCPTtranslatorRule>
        <SCPTdelayTime>0</SCPTdelayTime>
    </Item>
  </iLonItem>
</Set>
```

12.3.4.5 SNVT_scene_TO_SNVT_setting

You can use this Type Translator Rule to translate an input data point of type **SNVT_scene** to an output data point of type **SNVT_setting**.

- The **SNVT_scene** input data point is defined by a <DataPointFormat> element that has a “Dp Type” attribute of “Input”. The input data point referenced by <UCPTname> must have a <UCPTformatDescription> property of **#0000000000000000[0]. SNVT_scene** and a <UCPTnickName> property of Input0.
- The **SNVT_setting** output data point to be translated is defined by a <DataPoint> element that have a “Dp Type” attribute of “Output”. The output data point referenced by <UCPTname> must have a <UCPTformatDescription> of **#0000000000000000[0]. SNVT_setting** and a <UCPTnickName> of Output0.

Each time a type translation is made, the function, rotation, and setting fields of the **SNVT_setting** ouput data point are assigned values based on the current values stored in the scene_function and scene_number fields of the **SNVT_scene** input data point, as described in the following table:

SNVT_scene (input point)	**SNVT_setting (ouput point)**
function: SC_RECALL scene_number: <= 4	function: SET_STATE (enumerated value is 5) setting: <=25*scene_number> rotation: 0
function: SC_RECALL scene_number: >= 5	function: SET_NUL (enumerated value is -1) setting: 100 rotation: 0
function: SC_NUL scene_number: any	function: SET_NUL setting: 100 rotation: 0

The following code demonstrates how to use the *Set* function to create a type translator that uses this <UCPTtranslatorRule>:

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTtypeTranslator_Cfg">
        <UCPTname>Net/LON/iLON App/Type Translator[6]</UCPTname>
        <DataPoint dpType="Input">
            <UCPTname>Net/LON/iLON App/VirtFb/nviScene_1</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_scene</UCPTformatDescription>
            <UCPTnickName>Input0</UCPTnickName>
        </DataPoint>
            <DataPoint dpType="Output">
            <UCPTname>Net/LON/iLON App/VirtFb/nvoSetting_1</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_setting</UCPTformatDescription>
            <UCPTnickName>Output0</UCPTnickName>
        </DataPoint>
        <UCPTtranslatorRule>SNVT_scene_TO_SNVT_setting</UCPTtranslatorRule>
        <SCPTdelayTime>0</SCPTdelayTime>
    </Item>
  </iLonItem>
</Set>
```

12.3.4.6 SNVT_scene_TO_SNVT_switch

You can use this Type Translator Rule to translate an input data point of type **SNVT_scene** to an output data point of type **SNVT_switch**.

- The **SNVT_scene** input data point is defined by a <DataPointFormat> element that has a "Dp Type" attribute of "Input". The input data point referenced by <UCPTname> must have a <UCPTformatDescription> property of **#0000000000000000[0]. SNVT_scene** and a <UCPTnickName> property of Input0.
- The **SNVT_switch** output data point to be translated is defined by a <DataPoint> element that have a "Dp Type" attribute of "Output". The output data point referenced by <UCPTname> must have a <UCPTformatDescription> of **#0000000000000000[0]. SNVT_switch** and a <UCPTnickName> of Output0.

You can 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.

SNVT_scene (input point)	**SNVT_switch (ouput point)**
function: SC_NUL scene_number: 0	No update made to output data point
function: SC_NUL scene_number: >0	value: 0.0 state: 0 (OFF)
function: SC_RECALL scene_number: 1 function: SC_RECALL scene_number: 2 function: SC_RECALL scene_number: 3 function: SC_RECALL scene_number: >3 function: SC_RECALL scene_number: 255	value: 25.0 state: 1 (ON) value: 50.0 state: 1 (ON) value: 75.0 state: 1 (ON) value: 100.0 state: 1 (ON) value: 0.0 state: 0 (OFF)

The following code demonstrates how to use the *Set* function to create a type translator that uses this <UCPTtranslatorRule>:

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTtypeTranslator_Cfg">
        <UCPTname>Net/LON/iLON App/Type Translator[7]</UCPTname>
        <DataPoint dpType="Input">
            <UCPTname>Net/LON/iLON App/VirtFb/nviScene_1</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_scene</UCPTformatDescription>
            <UCPTnickName>Input0</UCPTnickName>
        </DataPoint>
            <DataPoint dpType="Output">
            <UCPTname>Net/LON/iLON App/VirtFb/nvoSwitch_1</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
            <UCPTnickName>Output0</UCPTnickName>
        </DataPoint>
        <UCPTtranslatorRule>SNVT_scene_TO_SNVT_switch</UCPTtranslatorRule>
        <SCPTdelayTime>0</SCPTdelayTime>
    </Item>
  </iLonItem>
</Set>
```

12.3.4.7 SNVT_setting_TO_SNVT_switch

You can use this Type Translator Rule to translate an input data point of type **SNVT_setting** to an output data point of type **SNVT_switch**.

- The **SNVT_setting** input data point is defined by a <DataPointFormat> element that has a "Dp Type" attribute of "Input". The input data point referenced by <UCPTname> must have a <UCPTformatDescription> property of **#0000000000000000[0]. SNVT_setting** and a <UCPTnickName> property of Input0.
- The **SNVT_switch** output data point to be translated is defined by a <DataPoint> element that have a "Dp Type" attribute of "Output". The output data point referenced by <UCPTname> must have a <UCPTformatDescription> of **#0000000000000000[0]. SNVT_switch** and a <UCPTnickName> of Output0.

Each time a type translation is made, the value and state fields of the **SNVT_switch** ouput data point are assigned values based on the current values stored in the function and setting fields of the **SNVT_setting** input data point, as described in the following table:

SNVT_setting (input point)	**SNVT_switch (ouput point)**
function: SET_STATE setting: <=100.0	value: 0.0 state: 0 (OFF)
function: SET_STATE setting: >100.0	value: <=setting> state: 0 (OFF)
function: SET_NUL setting: any	value: 0.0 state: 0 (OFF)

The following code demonstrates how to use the *Set* function to create a type translator that uses this <UCPTtranslatorRule>:

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTtypeTranslator_Cfg">
        <UCPTname>Net/LON/iLON App/Type Translator[8]</UCPTname>
        <DataPoint dpType="Input">
            <UCPTname>Net/LON/iLON App/VirtFb/nviSetting_1</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_setting</UCPTformatDescription>
            <UCPTnickName>Input0</UCPTnickName>
        </DataPoint>
            <DataPoint dpType="Output">
            <UCPTname>Net/LON/iLON App/VirtFb/nvoSwitch_1</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
            <UCPTnickName>Output0</UCPTnickName>
        </DataPoint>
```

```
        <UCPTtranslatorRule>SNVT_setting_TO_SNVT_switch</UCPTtranslatorRule>
        <SCPTdelayTime>0</SCPTdelayTime>
    </Item>
  </iLonItem>
</Set>
```

12.3.4.8 SNVT_state_TO_16xSNVT_switch

You can use this Type Translator Rule to convert a single **SNVT_state** input data point to up to 16 **SNVT_switch** output data points. 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 input data point is defined by a <DataPointFormat> element that have a "Dp Type" attribute of "Input". The input data point referenced by <UCPTname> must have a <UCPTformatDescription> property of **#0000000000000000[0]. SNVT_state** and a <UCPTnickName> of "Input0".
- The 16 **SNVT_switch** data points to be translated are defined by a list of <DataPoint> elements that have a "Dp Type" attribute of "Output". The output data points referenced by <UCPTname> must have a <UCPTformatDescription> property of **#0000000000000000[0]. SNVT_switch** and a <UCPTnickName> in the range of Output0 to Output15.

The value of the *state* field of each input data point will be read and stored in bitX of the output data point, where X represents the <UCPTnickName> of the input data point. The state field in each **SNVT_switch** output data point will be assigned a value based on the <UCPTnickName> of the output data point and the corresponding value in bitX of the **SNVT_state** input data point. For example, the **SNVT_switch** output data point with a <UCPTnickName> of Output0 will be assigned a value based on *Bit0* of the input data point. Or, the **SNVT_switch** output data point with a <UCPTnickName> of Output8 will be assigned a value based on *Bit8* of the input data point.

If the value of a BitX field of the **SNVT_state** input data point is 0, then the applicable **SNVT_switch** output data point will be assigned the value 0.0 0. If the value of a BitX field of the **SNVT_state** input data point is 1, then the applicable **SNVT_switch** output data point will be assigned the value 100.0 1.

The following code demonstrates how to use the *Set* function to create a type translator that uses this <UCPTtranslatorRule>:

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTtypeTranslator_Cfg">
        <UCPTname>Net/LON/iLON App/Type Translator[10]</UCPTname>
        <DataPoint dpType="Input">
            <UCPTname>Net/LON/iLON App/VirtFb/nvoState</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_state</UCPTformatDescription>
            <UCPTnickName>Input0</UCPTnickName>
        </DataPoint>
        <DataPoint dpType="Output">
            <UCPTname>Net/LON/iLON App/VirtFb/nviSwitch_1</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
            <UCPTnickName>Output0</UCPTnickName>
        </DataPoint>
        <DataPoint dpType="Output">
            <UCPTname>Net/LON/iLON App/VirtFb/nviSwitch_2</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
            <UCPTnickName>Output1</UCPTnickName>
        </DataPoint>
        <UCPTtranslatorRule>SNVT_state_TO_16xSNVT_switch</UCPTtranslatorRule>
        <SCPTdelayTime>0</SCPTdelayTime>
    </Item>
 </iLonItem>
</Set>
```

12.3.4.9 SNVT_switch_TO_SNVT_lev_disc

You can 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**.

- The **SNVT_switch** input data point is defined by a <DataPointFormat> element that has a "Dp Type" attribute of "input". The input data point referenced by <UCPTname> must have a <UCPTformatDescription> property of **#0000000000000000[0]. SNVT_switch** and a <UCPTnickName> property of Input0.
- The **SNVT_lev_disc** output data point to be translated is defined by a <DataPoint> element that have a "Dp Type" attribute of "Output". The output data point referenced by <UCPTname> must have a <UCPTformatDescription> of **#0000000000000000[0]. SNVT_lev_disc** and a <UCPTnickName> of Output0.

Each time a type translation is made, the **SNVT_lev_desc** output data point is assigned an enumeration based on the current value and state stored in the **SNVT_switch** input data point, as described in the following table:

SNVT_switch (input point)	SNVT_lev_desc (output point)
value: any state: 0	ST_NUL
value: 0.0 state: 1	ST_OFF
value: 0.1–25.0 state: 1	ST_LOW
value: 25.0–50.0 state: 1	ST_MED
value: 50.0–75.0 state: 1	ST_HIGH
value: 75.0–100.0 state: 1	ST_ON
value: >100.0 state: 1	ST_NUL

The following code demonstrates how to use the *Set* function to create a type translator that uses this <UCPTtranslatorRule>:

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTtypeTranslator_Cfg">
        <UCPTname>Net/LON/iLON App/Type Translator[11]</UCPTname>
        <DataPoint dpType="Input">
            <UCPTname>Net/LON/iLON App/VirtFb/nviSwitch</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_switch </UCPTformatDescription>
            <UCPTnickName>Input0</UCPTnickName>
        </DataPoint>
            <DataPoint dpType="Output">
            <UCPTname>Net/LON/iLON App/VirtFb/nvolevDisc</UCPTname>
            <UCPTformatDescription>#0000000000000000[0].SNVT_lev_disc</UCPTformatDescription>
            <UCPTnickName>Output0</UCPTnickName>
        </DataPoint>
        <UCPTtranslatorRule>SNVT_switch_TO_SNVT_lev_disc</UCPTtranslatorRule>
        <SCPTdelayTime>0</SCPTdelayTime>
    </Item>
  </iLonItem>
</Set>
```

12.3.5 Using the Delete Function on a Type Translator

You can use the *Delete* function to delete a Type Translator. To delete a Type Translator, you provide an <Item> element with a UFPTtypeTranslator_Cfg type that includes the <UCPTname> property of the type translator to be deleted. The following code sample demonstrates how to use the *Delete* function to delete a Type Translator:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTtypeTranslator_Cfg">
        <UCPTname>Net/LON/iLON App/Type Translator[1]</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/LON/iLON App/Type Translator[1]</UCPTname>
    </Item>
  </iLonItem>
</DeleteResponse>
```

13 Type Translator Rules

You can use the Type Translator Rule SOAP functions to create additional Type Translator Rules for the SmartServer, or to modify the Type Translator Rules provided with the SmartServer software. Each Type Translator Rule defines the network variable type of the data points a Type Translator will accept 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 will 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. In addition, this chapter describes the pre-defined type translator rules included with the SmartServer software. For more information on the Type Translator application, see Chapter 12, *Type Translator*.

13.1 Type Translator Rule XML Files

The configuration of each Type Translator Rule defined for the SmartServer will be stored in an XML file in the `/root/config/Software/translatorRules` directory of the SmartServer. 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.

This chapter describes how to use the SOAP interface to create a new Type Translator Rule, how to modify an existing Type Translator Rule, and how to use the pre-defined Type Translator Rules on the SmartServer.

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 two **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.

```
<Item xsi:type="UFPTtypeTranslator_Rule_Cfg" >
    <UCPTname>2xSwitch_to_Switch</UCPTname>
    <UCPTannotation>8000010128000000[4].UFPTtypeTranslator_Rule</UCPTannotation>
    <UCPThidden>0</UCPThidden>
    <UCPTlastUpdate>2008-03-13T13:47:47.330-07:00</UCPTlastUpdate>
    <UCPTdescription>(New Rule)</UCPTdescription>
    <UCPTuri>#8000010128000000[4].UFPTtypeTranslator_Rule_Cfg.htm</UCPTuri>
    <DataPointFormat>
        <UCPTnickName>nviSwitch1</UCPTnickName>
        <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
    </DataPointFormat>
    <DataPointFormat>
        <UCPTnickName>nviSwitch2</UCPTnickName>
```

```
        <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
    </DataPointFormat>
    <DataPointFormat>
        <UCPTnickName>nvoSwitch</UCPTnickName>
        <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
    </DataPointFormat>
    <Case>
        <UCPTindex>0</UCPTindex>
        <UCPTcompFunction LonFormat="UCPTcompFunction">FN_NUL</UCPTcompFunction>
        <UCPTcompValue LonFormat="">0</UCPTcompValue>
        <Rule>
            <UCPTindex>0</UCPTindex>
            <UCPTinputPath>DataPointFormat[UCPTnickName=“nviSwitch1”]</UCPTinputPath>
            <UCPTinputFieldName>state</UCPTinputFieldName>
            <UCPToutputPath>DataPointFormat[UCPTnickName=“nvoSwitch”]</UCPToutputPath>
            <UCPToutputFieldName>state</UCPToutputFieldName>
            <UCPTcompFunction LonFormat="UCPTcompFunction">FN_NUL</UCPTcompFunction>
        </Rule>
    </Case>
    <Case>
        <UCPTindex>1</UCPTindex>
        <UCPTinputPath>DataPointFormat[UCPTnickName=“nviSwitch1”]</UCPTinputPath>
        <UCPTcompFunction LonFormat="UCPTcompFunction">FN_NUL</UCPTcompFunction>
        <UCPTcompValue LonFormat="">0</UCPTcompValue>
        <Rule>
            <UCPTindex>0</UCPTindex>
            <UCPTinputPath>DataPointFormat[UCPTnickName=“nviSwitch2”]</UCPTinputPath>
            <UCPTinputFieldName>value</UCPTinputFieldName>
            <UCPToutputPath>DataPointFormat[UCPTnickName=“nvoSwitch”]</UCPToutputPath>
            <UCPToutputFieldName>value</UCPToutputFieldName>
            <UCPTcompFunction LonFormat="UCPTcompFunction">FN_NUL</UCPTcompFunction>
        </Rule>
    </Case>
</Item>
```

13.2 Creating and Modifying the Type Translator Rule XML Files

You can create and manage the Type Translator rules with the *Set* SOAP function. The following section, *Type Translator Rule SOAP Interface*, describes how to use *Set* 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 SmartServer via FTP. Echelon does not recommend this, as the SmartServer will require a reboot to read the configuration of the downloaded file. Additionally, the SmartServer 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.

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’s configuration.

13.3 Type Translator Rule SOAP Interface

You can use the SOAP interface to perform the following functions on a type translator rule:

Function	Description
List	Generate a list of the type translator rules on the SmartServer.
Get	Retrieve the configuration of the type translator rules on the SmartServer.
Set	Create a new type translator rule, or overwrite the configuration of an existing type translator rule.

Delete	Delete a type translator rule.

13.3.1 Using the List Function on a Type Translator Rule

You can use the *List* function to retrieve a list of the Type Translator Rules that you have added to the SmartServer. The List function takes an <iLonItem> element that includes an xSelect statement querying items of a UFPTtypeTranslator_Rule_Cfg type as its input, as shown in the example below. The *List* function returns an <Item> element for each type translator rule on the SmartServer included in the specified xSelect statement. The next section describes the properties included in each of these elements.

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="UFPTtypeTranslator_Rule_Cfg"]</xSelect>
  </iLonItem>
</List>
```

Response

```
<ListResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
    <Item>
        <UCPTname>My Custom Rule: SNVT_temp to SNVT_temp_p</UCPTname>
        <UCPTannotation>8000010128000000[4].UFPTtypeTranslator_Rule;
         xsi:type="UFPTtypeTranslator_Rule_Cfg"
        </UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
    <Item>
        <UCPTname>My Custom Rule: 2xSwitch_to_Switch</UCPTname>
        <UCPTannotation>8000010128000000[4].UFPTtypeTranslator_Rule;
          xsi:type="UFPTtypeTranslator_Rule_Cfg"
        </UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
  </iLonItem>
</ListResponse>
```

13.3.2 Using the Get Function on a Type Translator Rule

You can use the *Get* function to retrieve the configuration of any type translator rule on the SmartServer. You must reference the type translator whose configuration is to be returned by its <UCPTname> in the input you supply to the function, as shown in the example below.

Request

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>My Custom Rule: 2xSwitch_to_Switch</UCPTname>
    </Item>
  </iLonItem>
</Get>
```

Response

```
<GetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="UFPTtypeTranslator_Rule_Cfg" >
        <UCPTname>2xSwitch_to_Switch</UCPTname>
        <UCPTannotation>8000010128000000[4].UFPTtypeTranslator_Rule</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-13T13:47:47.330-07:00</UCPTlastUpdate>
```

```
        <UCPTdescription>Translates the state and value fields from two SNVT_switch input DPs to one
                         output SNVT_swicth DP
        </UCPTdescription>
        <UCPTuri>#8000010128000000[4].UFPTtypeTranslator_Rule_Cfg.htm</UCPTuri>
        <DataPointFormat>
            <UCPTnickName>nviSwitch1</UCPTnickName>
            <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
        </DataPointFormat>
        <DataPointFormat>
            <UCPTnickName>nviSwitch2</UCPTnickName>
            <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
        </DataPointFormat>
        <DataPointFormat>
            <UCPTnickName>nvoSwitch</UCPTnickName>
            <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
        </DataPointFormat>
        <Case>
            <UCPTindex>0</UCPTindex>
            <UCPTcompFunction LonFormat="UCPTcompFunction">FN_NUL</UCPTcompFunction>
            <UCPTcompValue LonFormat="">0</UCPTcompValue>
            <Rule>
                <UCPTindex>0</UCPTindex>
                <UCPTinputPath>DataPointFormat[UCPTnickName=“nviSwitch1”]</UCPTinputPath>
                <UCPTinputFieldName>state</UCPTinputFieldName>
                <UCPToutputPath>DataPointFormat[UCPTnickName=“nvoSwitch”]</UCPToutputPath>
                <UCPToutputFieldName>state</UCPToutputFieldName>
                <UCPTcompFunction LonFormat="UCPTcompFunction">FN_NUL</UCPTcompFunction>
            </Rule>
        </Case>
        <Case>
            <UCPTindex>1</UCPTindex>
            <UCPTinputPath>DataPointFormat[UCPTnickName=“nviSwitch1”]</UCPTinputPath>
            <UCPTcompFunction LonFormat="UCPTcompFunction">FN_NUL</UCPTcompFunction>
            <UCPTcompValue LonFormat="">0</UCPTcompValue>
            <Rule>
                <UCPTindex>0</UCPTindex>
                <UCPTinputPath>DataPointFormat[UCPTnickName=“nviSwitch2”]</UCPTinputPath>
                <UCPTinputFieldName>value</UCPTinputFieldName>
                <UCPToutputPath>DataPointFormat[UCPTnickName=“nvoSwitch”]</UCPToutputPath>
                <UCPToutputFieldName>value</UCPToutputFieldName>
                <UCPTcompFunction LonFormat="UCPTcompFunction">FN_NUL</UCPTcompFunction>
            </Rule>
        </Case>
    </Item>
  </iLonItem>
</GetResponse>
```

The function returns an <Item> element for each type translator rule referenced in the input parameters supplied to the function. The properties included in each element are initially defined when the type translator rule is created. You can write to these properties with the *Set* function. The following table describes these properties.

Property	Description
`<UCPTname>`	The name of the type translator rule in the following format: *<network/channel/device/functional block>*.
`<UCPTannotation>`	The program ID and functional profile template used by the type translator rule. This property is always `8000010128000000[4].UFPTtypeTranslator_Rule`
`<UCPThidden>`	A flag indicating whether the type translator rule functional block is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values:

Property	Description
	0 – shown **1** – hidden
<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**
<UCPTuri>	The name of the file containing the configuration web page for the Type Translator Rule on the SmartServer flash disk, absolute or relative to /web/user/echelon. This property is #8000010128000000[4].UFPTtypeTranslator_Rule_Cfg.htm by default.
<UCPTdescription>	A description of the Type Translator Rule. This can be a maximum of 227 characters long.
<DataPointFormat>	You can define the input and output data points that a Type Translator Rule accepts with a series of <DataPointFormat> elements. Each <DataPointFormat> element is a structure that contains two properties: <UCPTnickName> and <UCPTformatDescription>. • <UCPTnickName>. When you create a Type Translator to use a rule, you can assign each data point a nickname. that is used to reference the data point. You can use this property to reference a data point in the <InputPath> and <OutputPath> properties in a type translator rule. By default, the nickname of the data point is the data point's name. For example, a data point with a <UCPTname> of "Net/LON/iLON App/VirtFb/nviSwitch2" has a default <UCPTnickname> of "nviSwitch2". • <UCPTformatDescription>. The data point's program ID; data type (SNVT, SCPT, UNVT, UCPT, or built-in data type); and format (e.g., SI metric or US customary if the type has multiple formats such as **SNVT_temp_p**). The format description is displayed in the following format: • **#**<manufacturer ID>**[**scope selector**].**<type name>[#format] .
<Case>	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 to and from scalar, structured, and enumerated data point values. 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*.

13.3.2.1 Creating a Case Structure

You can create case structures for each Type Translator Rule that defines the set of operations that will be performed when a type translation is made with that rule. 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. See *Case Rules* for more information on the <Rule> element.

For example, consider a Type Translator Rule that converts a **SNVT_occupancy** input data point (nviOccupancy) to a **SNVT_switch** output data point (nvoSwitch). You could set up one case structure that sets nvoSwitch to ON (100.0 1) when nviOccupancy is set to OC_OCCUPIED. You could set up another case structure that sets nvoSwitch to OFF (0.0 0) when nviOccupancy 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.

```
<Case>
    <UCPTindex>0</UCPTindex>
    <UCPTinputPath>DataPointFormat[UCPTnickName="nviOccupancy"]</UCPTinputPath>
    <UCPTcompFunction LonFormat="UCPTcompFunction">FN_EQ</UCPTcompFunction>
    <UCPTcompValue LonFormat="">OC_OCCUPIED</UCPTcompValue>
    <Rule>
        <UCPTindex>0</UCPTindex>
        <UCPTinputPath>DataPointFormat[UCPTnickName="nviOccupancy"]</UCPTinputPath>
        <UCPToutputPath>DataPointFormat[UCPTnickName="nvoSwitch"]</UCPToutputPath>
        <UCPToutputFieldName />
        <UCPTcompFunction LonFormat="UCPTcompFunction">FN_NUL</UCPTcompFunction>
        <UCPTmultiplier>0</UCPTmultiplier>
        <UCPTconstant LonFormat="">100.0 1</UCPTconstant>
    </Rule>
</Case>
<Case>
    <UCPTindex>1</UCPTindex>
    <UCPTinputPath>DataPointFormat[UCPTnickName="nviOccupancy"]</UCPTinputPath>
    <UCPTcompFunction LonFormat="UCPTcompFunction">FN_EQ</UCPTcompFunction>
    <UCPTcompValue LonFormat="">OC_UNOCCUPIED</UCPTcompValue>
    <Rule>
        <UCPTindex>0</UCPTindex>
        <UCPTinputPath>DataPointFormat[UCPTnickName="nviOccupancy"]</UCPTinputPath>
        <UCPToutputPath>DataPointFormat[UCPTnickName="nvoSwitch"]</UCPToutputPath>
        <UCPToutputFieldName />
        <UCPTcompFunction LonFormat="UCPTcompFunction">FN_NUL</UCPTcompFunction>
        <UCPTmultiplier>0</UCPTmultiplier>
        <UCPTconstant LonFormat="">0.0 0</UCPTconstant>
    </Rule>
</Case>
```

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.

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.

The following table 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.

Property	Description
<UCPTindex>	The index number of the case structure.
<UCPTinputPath>	If you are using an existing type translator rule (a pre-defined type translator rule on the SmartServer or a custom rule you previously created), you need to specify the <UCPTnickName> property of the input point's DataPointFormat element using the following format: <UCPTinputPath> DataPointFormat[UCPTnickName="Input0"] </UCPTinputPath> You do not need to specify this property if you are creating a custom type translator rule.
<UCPTinputValue>	If you are using an existing type translator rule on the SmartServer and the specified input point has a structured data type, you need to specify the field to be translated. For example, you would use the following code to specify that the value field of a **SNVT_switch** input point is to be translated an output point: <UCPTinputFieldName>value</UCPTinputFieldName> You do not need to specify this property if you are creating a custom type translator rule.
<UCPTcompFunction>	Select a comparison function (UCPTcompFunction) for the case. The following lists and describes the comparison functions that can be used to fill in the <UCPTcompFunction> property. FN_GT — Greater than. Returns True if the value of the input data point is greater than that of the compare data point. 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. The value of the input data point, or input data point field, selected for the case structure 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 on case rules, see *Case Rules*.
<UCPTcompValue>	Specify a value to be compared against the value of the selected data point or data point field. This property is not used if <UCPTcompFunction> is FN_NUL.

13.3.2.2 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 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.

For example, consider a Type Translator Rule that converts a **SNVT_scene** data point (nviScene) to a **SNVT_switch** data point (nvoSwitch). 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.

```
<Case>
    <UCPTindex>0</UCPTindex>
    <UCPTinputPath>DataPointFormat[UCPTnickName="nviScene"]</UCPTinputPath>
    <UCPTinputFieldName>scene_number</UCPTinputFieldName>
    <UCPTcompFunction LonFormat="UCPTcompFunction">FN_GT</UCPTcompFunction>
    <UCPTcompValue LonFormat="">2</UCPTcompValue>
    <Rule>
        <UCPTindex>0</UCPTindex>
        <UCPTinputPath>DataPointFormat[UCPTnickName="nviScene"]</UCPTinputPath>
        <UCPTinputFieldName>scene_number</UCPTinputFieldName>
        <UCPToutputPath>DataPointFormat[UCPTnickName="nvoSwitch"]</UCPToutputPath
        <UCPTcompFunction LonFormat="UCPTcompFunction">FN_NE</UCPTcompFunction>
        <UCPTcompValue LonFormat="">SC_RESET</UCPTcompValue>
        <UCPTmultiplier>0</UCPTmultiplier>
        <UCPTconstant LonFormat="">0.0 0</UCPTconstant>
    </Rule>
</Case>
<Case>
    <UCPTindex>1</UCPTindex>
    <UCPTinputPath>DataPointFormat[UCPTnickName="nviScene"]</UCPTinputPath>
    <UCPTinputFieldName>scene_number</UCPTinputFieldName>
    <UCPTcompFunction LonFormat="UCPTcompFunction">FN_LT</UCPTcompFunction>
```

```
    <UCPTcompValue LonFormat="">2</UCPTcompValue>
    <Rule>
        <UCPTindex>0</UCPTindex>
        <UCPTinputPath>DataPointFormat[UCPTnickName="nviScene"]</UCPTinputPath>
        <UCPTinputFieldName>scene_number</UCPTinputFieldName>
        <UCPToutputPath>DataPointFormat[UCPTnickName="nvoSwitch"]</UCPToutputPath>
        <UCPTcompFunction LonFormat="UCPTcompFunction">FN_NUL</UCPTcompFunction>
        <UCPTmultiplier>0</UCPTmultiplier>
        <UCPTconstant LonFormat="">100.0 1</UCPTconstant>
    </Rule>
</Case>
```

Each case rule is defined by a <Rule> element. The following table describes the properties that should be filled in within each <Rule> element to define each case rule.

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 (i.e. the one with the highest index number) takes precedence.
<UCPTinputPath>	A reference to the <UCPTnickName> property of the input data point. 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 specified in the <UCPToutputPath> property.
<UCPTinputValue>	If the input point has a structured data type and you need to evaluate a field within the structure, you need to specify that field in this property. You do not need to specify this property if you are evaluating a scalar data point or a structured data point as a whole.
<UCPToutputPath>	A reference to the <UCPTnickName> property of the output data point in which the value calculated by this case rule is to be stored.
<UCPToutputValue>	If the output point has a structured data type and you are writing to a field within the structure, you need to specify that field in this property. You do not need to specify this property if you are writing to a scalar data point or a structured data point as a whole.
<UCPTcompFunction> <UCPTcompValue>	If you are evaluating a data point in the case rule, specify the <UCPTcompFunction> and <UCPTcompValue> properties. If you are not evaluating a data point in the rule, specify FN_NUL for <UCPTcompFunction>. You do not need to specify a <UCPTcompValue> property in this case. The <UCPTcompValue> selected must use the same value format

Property	Description
	as the input data point or field selected for the case rule. The following table lists and describes the comparison functions you can use to fill in the <UCPTcompFunction> property. FN_GT — Greater than. Returns True if the value of the input data point is greater than that of the compare data point. 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. The value of the input data point or field will be compared to the compare value using the 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.
<UCPTmultiplier>	If the output data point or 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 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.

Property	Description
<UCPTconstant>	If the output data point or 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 field takes a numeric value as its value type, enter a numeric value here. The Type Translator will add this to the value of the input data point (or data point field) 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.

13.3.3 Using the Set Function on a Type Translator Rule

You can use the *Set* function to create new type translator rules, or to overwrite the configuration of existing type translator rule. The type translator rules to be created or written are signified by a list of <Item> elements in the input you supply to the function. The properties you must define within each <Item> element are the same, whether you are creating a new type translator or modifying an existing type translator. The previous section, *Using the Get Function on a Type Translator Rule*, describes these properties.

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 SmartServer. Once the file has been generated, you can reference the rule when creating a Type Translator, as described in Chapter 12.

Notes: If you specify a type translator rule with the <UCPTname> element, the *Set* function deletes the specified type translator rule before the specified parameters are set. If the <UCPTname> element is not specified, a new type translator rule is created.

When modifying an existing type translator rule, any optional properties omitted from the input will be erased. Old values will not be preserved, so you should 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 rule with this function, you may want to use output from the *Get* function as the basis for your input. 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 following example uses the *Set* function to create a Type Translator Rule definition that translates a **SNVT_switch** data point (nviSwitch) to a **SNVT_hvac_mode** data point (nvoHVACmode).

Request

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTtypeTranslator_Rule_Cfg">
        <UCPTname>My Custom Rule: SNVT_switch to SNVT_hvac_mode</UCPTname>
        <UCPTdescription>converts SNVT_switch ON to SNVT_hvac_mode HVAC_COOL</UCPTdescription>
        <DataPointFormat>
            <UCPTnickName>nviSwitch</UCPTnickName>
            <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
        </DataPointFormat>
        <DataPointFormat>
            <UCPTnickName>nvoHVACmode</UCPTnickName>
            <UCPTformatDescription>#0000000000000000[0].SNVT_hvac_mode</UCPTformatDescription>
        </DataPointFormat>
        <Case>
            <UCPTindex>0</UCPTindex>
            <UCPTinputPath>DataPointFormat[UCPTnickName="nviSwitch"]</UCPTinputPath>
            <UCPTcompFunction LonFormat="UCPTcompFunction">FN_EQ</UCPTcompFunction>
            <UCPTcompValue LonFormat="">100.0 1</UCPTcompValue>
```

```
            <Rule>
                <UCPTindex>0</UCPTindex>
                <UCPTinputPath>DataPointFormat[UCPTnickName="nviSwitch"]</UCPTinputPath>
                <UCPToutputPath>DataPointFormat[UCPTnickName="nvoHVACmode"]</UCPToutputPath>
                <UCPTcompFunction LonFormat="UCPTcompFunction">FN_NUL</UCPTcompFunction>
                <UCPTmultiplier>0</UCPTmultiplier>
                <UCPTconstant LonFormat="">HVAC_COOL</UCPTconstant>
            </Rule>
        </Case>
        <Case>
            <UCPTindex>1</UCPTindex>
            <UCPTinputPath>DataPointFormat[UCPTnickName="nviSwitch"]</UCPTinputPath>
            <UCPTcompFunction LonFormat="UCPTcompFunction">FN_EQ</UCPTcompFunction>
            <UCPTcompValue LonFormat="">0.0 0</UCPTcompValue>
            <Rule>
                <UCPTindex>0</UCPTindex>
                <UCPTinputPath>DataPointFormat[UCPTnickName="nviSwitch"]</UCPTinputPath>
                <UCPToutputPath>DataPointFormat[UCPTnickName="nvoHVACmode"]</UCPToutputPath>
                <UCPTcompFunction LonFormat="UCPTcompFunction">FN_NUL</UCPTcompFunction>
                <UCPTmultiplier>0</UCPTmultiplier>
                <UCPTconstant LonFormat="">HVAC_OFF</UCPTconstant>
            </Rule>
        </Case>
    </Item>
  </iLonItem>
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>My Custom Rule: SNVT_switch to SNVT_hvac_mode</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

13.3.4 Using the Delete Function on a Type Translator Rule

You can use the *Delete* function to delete a Type Translator Rule. To delete a Type Translator Rule, you provide an <Item> element with a UFPTtypeTranslator_Rule_Cfg type that includes the <UCPTname> property of the type translator rule to be deleted. The following code sample demonstrates how to use the *Delete* function to delete a Type Translator Rule:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="UFPTtypeTranslator_Rule_Cfg">
        <UCPTname>My Custom Rule: SNVT_switch to SNVT_hvac_mode</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>My Custom Rule: SNVT_switch to SNVT_hvac_mode</UCPTname>
    </Item>
  </iLonItem>
</DeleteResponse>
```

14 LONWORKS Driver

The following chapter describes how to manage the networks, channels, devices, functional blocks, and data points on the LONWORKS driver on the SmartServer. Note that the functions and properties listed in this chapter are applicable for both the Web service on the SmartServer and the LNS Proxy Web service.

14.1 LONWORKS Networks

The following section describes how to use the *List*, *Get*, *Set*, and *Delete* functions on LONWORKS networks.

14.1.1 Using the List Function on a LONWORKS Network

You can use the *List* function to retrieve the network connected to the SmartServer or a list of the LNS network databases in a specific LNS Server computer through the LNS Proxy Web service. The List function takes an <iLonItem> element that has an xSelect statement with a LON_Network_Cfg or Network_Cfg type as its input, as shown in the example below.

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="LON_Network_Cfg"]</xSelect>
  </iLonItem>
</List>
```

Response

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
    <Item>
        <UCPTname>MyNetwork</UCPTname>
        <UCPTannotation>iLonNS;xsi:type="LON_Network_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
  </iLonItem>
</List>
```

The *List* function returns an <Item> element for the network connected to the SmartServer or a list of <Item> elements for the LNS network databases in an LNS Server computer.

You could use the list of <Item> elements returned by this function as input for the *Get* function. The *Get* function would then return the configuration of each network included in the list. The next section describes the properties included in each of these elements.

14.1.2 Using the Get Function on a LONWORKS Network

You can use the *Get* function to retrieve the configuration of the network connected to the SmartServer or the LNS network databases stored in an LNS Server computer. The input parameters you supply to this function will include one or more <Item> elements with a LON_Network_Cfg type. Each <Item> element will include the <UCPTname> of each network whose configuration is to be returned by this function, as shown in the example below.

Note: If you omit the xSelect statement, the generic Network_Cfg item is returned also.

Request

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Network_Cfg">
        <UCPTname>MyNetwork</UCPTname>
    </Item>
  </iLonItem>
```

```
</Get>
```

Response

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="LON_Network_Cfg">
        <UCPTname>MyNetwork</UCPTname>
        <UCPTannotation>iLonNS;xsi:type="LON_Network_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-20T15:26:55.610-07:00</UCPTlastUpdate>
        <UCPTuri>LON_Network_Cfg.htm</UCPTuri>
        <UCPThandle>873906929</UCPThandle>
        <LnsDatabase>
            <UCPTname>MyNetwork</UCPTname>
            <UCPTservicePath>//WebService[UCPTindex=4]</UCPTservicePath>
        </LnsDatabase>
        <UCPTlnsSync LonFormat="UCPTlnsSync">SYNC_NUL</UCPTlnsSync>
        <UCPTmgmtMode LonFormat="UCPTmgmtMode">LCA_ONNET</UCPTmgmtMode>
        <UCPTlnsNetworkInterface>X.Default.SmartServer_RNI</UCPTlnsNetworkInterface>
        <Domain>
            <UCPTdomainIndex>0</UCPTdomainIndex>
            <UCPTdomainLength>1</UCPTdomainLength>
            <UCPTdomainKey>C0</UCPTdomainKey>
        </Domain>
        <Domain>
            <UCPTdomainIndex>1</UCPTdomainIndex>
            <UCPTdomainLength>0</UCPTdomainLength>
            <UCPTdomainKey />
        </Domain>
    </Item>
  </iLonItem>
</Get>
```

The *Get* function returns an <Item> element for each network referenced in the input parameters you supplied to the function. The properties included within each <Item> element are initially defined when the network is added to the SmartServer or LNS network database. You can write to these network properties with the *Set* function. The following table describes these properties.

Property	Description
<UCPTname>	The name of the network in the following format: <*network* >. You can only rename a network if <UCPTlnsSync> is set to SYNC_STANDALONE. To rename a network, provide the <UCPThandle> of the network and the desired <UCPTname> property. You cannot rename networks if <UCPTlnsSync> is set to SYNC_NUL or SYNC_LNS. This means you cannot rename networks stored in an LNS network database via the LNS Proxy Web service.
<UCPTannotation>	The type of object and its xsi type, which is LON_Network_Cfg. This determines the icon used to represent the network in the SmartServer Web interface. For the network on the SmartServer this property is " iLONNS " if the network database was created with the SmartServer, or it is "LNS" if the network was created with an LNS application such as the LonMaker tool.
<UCPThidden>	A flag indicating whether the network is hidden or shown in the navigation pane on the left side of the SmartServer Web

Property	Description
	interface. This property may have the following values: **0** – shown **1** – hidden
<UCPTlastUpdate>	A timestamp indicating the last time the configuration of the network was updated. This timestamp uses the ISO 8601 format, which is as follows: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Data Point was last updated. The second segment (after the **T):** HH:MM:SS.sss represents the time of day the configuration of the Data Point 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 be expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. It can be left out. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002. If it is not UTC, time shift has to be defined: 2008-02-28T09:59:53.660+01:00
<UCPTdescription>	A user-defined description of the network. This can be a maximum of 201 characters long.
<UCPTuri>	The name of the file on the SmartServer flash disk containing the configuration of the network. This property is always LON_Network_Cfg.htm.
<UCPThandle>	The handle of the network assigned by the LNS server.
<LnsDatabase>	The LNS network database associated with the network. This property is applicable for a network on the SmartServer that is synchronized with an LNS network database, or an LNS network database stored in an LNS Server computer that is accessible via the LNS Proxy Web service. This element has the following properties: • <UCPTname>. The name of the network in following format: <*network* >. This property may be a maximum of 14 characters, and it may include embedded spaces. • <UCPTservicePath>. A reference to the Web service path of the LNS Server where SOAP calls are transmitted in the following format: //WebService[UCPTindex=*x*]. The *x* refers to the index assigned to the LNS Server when it was added to the LAN.

Property	Description
<UCPTlnsSync>	Specifies the network management service used to manage the network. This property can be one of the following values: • **SYNC_STANDALONE**. The SmartServer is the network manager. It transmits all network management commands to the devices attached to its channel, and network configuration changes are stored in the internal SmartServer database. In standalone mode, the network functions as a master-slave system, where the SmartServer is the master to the slave devices. You can use standalone mode to operate a small, single-channel network that does not require LNS services, LONWORKS connections, or connections to other network management tools. FT-10 networks running in standalone mode are limited to a maximum of 64 devices; PL-20 networks are limited to a maximum of 200 devices. • **SYNC_LNS**. Network messages are routed through the LNS server specified in the <LnsDatabase> property. The SmartServer and the devices connected to it communicate in a peer-to-peer manner. In this mode, the SmartServer independently initiates communication with the LNS Proxy Web service, and the SmartServer automatically synchronizes with the LNS network database. You should use this mode as long as a firewall is not blocking the SmartServer's access to the port on the LNS Server computer selected for the LNS Proxy Web service (port 80 by default). • **SYNC_NUL**. Similar to **SYNC_LNS** except that the SmartServer has to be manually synchronized with the LNS server specified in the <LnsDatabase> property. This mode does not require the opening of any ports on firewalls blocking the SmartServer's access to the LNS Server computer. In this mode, the SmartServer Web interface serves as a proxy between the SmartServer and an LNS Server that is behind a firewall. When you manually synchronize the SmartServer to an LNS network database, the SmartServer Web interface requests a list of objects to be synced from the SmartServer via SOAP and forwards the objects returned by the SmartServer to the LNS Proxy Web service. The LNS Proxy Web service returns a set of synced objects to the SmartServer Web interface, which forwards these objects back to the SmartServer. You should only use this mode if a firewall is blocking the SmartServer's access to the LNS Proxy Web service port on the LNS Server computer (port 80 by default). This is the default.

Property	Description
<UCPTmgmtMode>	Specifies when network configuration changes are propagated over the network to devices. This property can be one of the following values: • **LCA_ONNET**. Changes are sent immediately to the devices on the network. Use **LCA_ONNET** if you are installing an engineered network, or if you are designing and installing an ad-hoc network at the same time. • **LCA_OFFNET.** Changes are stored in the network database and then sent to the devices on the network when you place the SmartServer OnNet. Use **LCA_OFFNET** if you are designing an engineered network.
<UCPTlnsNetworkInterface>	Specifies the network interface to be used for communication between the LNS Server and the network. To specify that the SmartServer is not attached to a network, you can omit this property in a *Set* function. This may be desired if you are performing network design tasks.
<UCPTdomain>	The domain length, index and key. If both the length and key are provided in a *Set* function, but they don't match, an error is thrown.

14.1.3 Using the Set Function on a LONWORKS Network

You can use the *Set* function to overwrite the configuration of a network, or to create a new network. The input parameters you supply to the function will include one or more <Item> elements. Each <Item> element includes a <UCPTname> property (case-insensitive) that specifies a unique network to be created or modified.

Each <Item> element may also include a series of properties that define the configuration of the new (or modified) network. This set of properties is the same, whether you are creating a new network or modifying an existing network. The previous section, *Using the Get Function on a LONWORKS Network*, details the properties you can include in the *Set* function.

You can set multiple networks with a single *Set* message. However, you should not attempt to create or write to more than 100 networks with a single call to the *Set* function.

Request (using the SmartServer Web service, create a new network and set it to Standalone mode)

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Network_Cfg">
        <UCPTname>MyNewNetwork_1</UCPTname>
        <UCPThandle xsi:type="number">873906929</UCPThandle>
        <LnsDatabase>
            <UCPTname>MyNewNetwork</UCPTname>
            <UCPTservicePath xsi:type="string">//WebService[UCPTindex=4]</UCPTservicePath>
        </LnsDatabase>
        <UCPTlnsSync xsi:type="string" LonFormat="UCPTlnsSync">SYNC_STANDALONE</UCPTlnsSync>
        <UCPTmgmtMode xsi:type="string" LonFormat="UCPTmgmtMode">LCA_ONNET</UCPTmgmtMode>
        <UCPTlnsNetworkInterface>X.Default.SmartServer_RNI</UCPTlnsNetworkInterface>
    </Item>
  </iLonItem>
</Set>
```

Note: In the *Set* function, you need to specify the <UCPThandle> property of the current network on the SmartServer. You can use the *List* and *Get* functions to get the network currently on the

SmartServer and get its <UCPThandle> property. If you don't specify the <UCPThandle> of the current network, you will receive a response message with the following fault structure:

```
<fault>
    <faultcode faultType="_error">8</faultcode>
    <faultstring xml:lang="en-US">It's not allowed to set more then one network.
    </faultstring>
</fault>
```

Request (using the SmartServer Web service, select an existing LNS network database to which the SmartServer will automatically be synchronized)

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="LON_Network_Cfg">
        <UCPTname>MyNewNetwork_1</UCPTname>
        <UCPThandle>873906929</UCPThandle>
        <LnsDatabase>
            <UCPTname>MyNewNetwork_1</UCPTname>
            <UCPTservicePath>//WebService[UCPTindex=4]</UCPTservicePath>
        </LnsDatabase>
        <UCPTlnsSync LonFormat="UCPTlnsSync">SYNC_LNS</UCPTlnsSync>
        <UCPTmgmtMode LonFormat="UCPTmgmtMode">LCA_ONNET</UCPTmgmtMode>
        <UCPTlnsNetworkInterface>X.Default.SmartServer_RNI</UCPTlnsNetworkInterface>
        </Item>
    </iLonItem>
</Set>
```

Note: You need to specify the <UCPThandle> property with the handle of the current network on the SmartServer or the response message will return an error.

Request (create a new network using the SmartServer Web service; use the same Set function to create the LNS network database via the LNS Proxy Web service and automatically synchronize the SmartServer to it)

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>MyNewNetwork_1</UCPTname>
        <UCPThandle xsi:type="number">873906929</UCPThandle>
        <LnsDatabase>
            <UCPTname>MyNewNetwork_1</UCPTname>
            <UCPTservicePath xsi:type="string">//WebService[UCPTindex=4]</UCPTservicePath>
        </LnsDatabase>
        <UCPTlnsSync xsi:type="string" LonFormat="UCPTlnsSync">SYNC_LNS</UCPTlnsSync>
        <UCPTmgmtMode xsi:type="string" LonFormat="UCPTmgmtMode">LCA_ONNET</UCPTmgmtMode>
        <UCPTlnsNetworkInterface>X.Default.SmartServer_RNI</UCPTlnsNetworkInterface>
    </Item>
  </iLonItem>
</Set>
```

Note: You need to specify the <UCPThandle> property with the handle of the current network on the SmartServer or the response message will return an error.

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>MyNewNetwork_1</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

14.1.3.1 Issuing Network Synchronization Commands

You can use the *InvokeCmd* function to manually synchronize the objects in a LONWORKS network to an LNS network database. If <UCPTlnsSync> is set to SYNC_NUL (LNS Manual; manual

synchronization), you can perform a manual synchronization to update the LNS network database with network configuration changes made with the SmartServer. Even if <UCPTlnsSync> is set to SYNC_LNS (LNS Auto; automatic synchronization), you may still want to periodically synchronize the SmartServer to update the SmartServer with changes made to the LNS network database by the LonMaker tool or other LNS application.

You can synchronize an entire LONWORKS network at one time, or you can select individual items to be synchronized. The input parameters you supply to this function include one <Item> element with a LON_Network_Command_Invoke type and a Command attribute that is set to "Syncronize" (misspelled intentionally). The <Item> element must include the <UCPTname> of the network object or objects being synchronized.

If you are synchronizing the entire network at once, you can synchronize only those objects in the network that have been modified in the SmartServer's internal database, or you can synchronize all objects regardless if they have been modified. To synchronize all the objects in the network, you insert a <UCPTannotation> element and set its *action* attribute to "SyncAll" in the <Item> element. The benefit of using the "SyncAll" option is that it synchronizes the LON driver properties of the objects if they have been changed in the SmartServer's internal database or the LNS network database. Examples of LON driver properties that you may want to keep synced include the timing parameters of a channel, the commission and application statuses of a device, and the format description of a data point.

If you are synchronizing individual channels, devices, or functional blocks, you can also synchronize the child objects of the specified objects. For example, when you synchronize a device, you can synchronize just that device, or you can synchronize the device and all of its child functional blocks and data points. To synchronize an object and all of its child objects, you insert a <UCPTannotation> element set its *action* attribute to "SyncRecursive" in the <Item> element.

Consider a scenario where you have multiple external devices on "Channel 1" of the "Building" network (Building/Channel 1/Device <*x*>, where *x* differentiates the devices).

- The following example demonstrates how to synchronize the network to the LNS network database (only objects that have been changed in the SmartServer's internal database):

```
<InvokeCmd xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Network_Command_Invoke" Command="Syncronize">
        <UCPTname>Building</UCPTname>
    </Item>
  </iLonItem>
</InvokeCmd>
```

- The following example demonstrates how to synchronize the network to the LNS network database (updating all objects and their LON driver properties):

```
<InvokeCmd xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Network_Command_Invoke" Command="Syncronize">
        <UCPTname>Building</UCPTname>
        <UCPTannotation>action="SyncAll"</UCPTannotation>
    </Item>
  </iLonItem>
</InvokeCmd>
```

- The following example demonstrates how to synchronize two devices on the network to the LNS network database (but no child functional blocks or data points):

```
<InvokeCmd xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Network_Command_Invoke" Command="Syncronize">
        <UCPTname>Building/Channel 1/Device 1</UCPTname>
    </Item>
    <Item xsi:type="LON_Network_Command_Invoke" Command="Syncronize">
        <UCPTname>Building/Channel 1/Device 2</UCPTname>
    </Item>
  </iLonItem>
```

```
</InvokeCmd>
```

- The following example demonstrates how to synchronize the two devices on the network and their child functional blocks and data points to the LNS network database:

```
<InvokeCmd xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Network_Command_Invoke" Command="Syncronize">
        <UCPTname>Building/Channel 1/Device 1</UCPTname>
        <UCPTannotation>action="SyncRecursive"</UCPTannotation>
    </Item>
    <Item xsi:type="LON_Network_Command_Invoke" Command="Syncronize">
        <UCPTannotation>action="SyncRecursive"</UCPTannotation>
    </Item>
  </iLonItem>
</InvokeCmd>
```

In the <Item> element, you can insert a <UCPTannotation> element with the following attributes specifying synchronization options:

Option	Description
`action`	Setting this attribute to “SyncAll” synchronizes all objects in the SmartServer’s internal database with the LNS network database, including hidden items (items with their <UCPThidden> flag set to 1). Objects are synchronized regardless if they have been modified in the SmartServer’s internal database. Selecting this option also synchronizes the LON driver properties of the objects if they have been changed in the SmartServer’s internal database or the LNS network database. Setting this attribute to "SyncRecursive" synchronizes the specified objects and all of their child objects in the SmartServer’s internal database with the LNS network database.
`;clearHiddenFbs`	Setting this attribute to "true" removes all functional blocks on the SmartServer that do not have a corresponding functional block shape in the LonMaker tool or other LNS application, and deletes their XML configuration files from the SmartServer’s internal database.
`;sync`	You can set this attribute to the following values: • abort. Stops the current synchronization operation. • restart. Stops and then re-starts the current synchronization operation. • state. Returns the number of remaining items to be synced.

The following example demonstrates how to synchronize a network with the synchronization options set in the <UCPTannotation> element:

```
<InvokeCmd xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Network_Command_Invoke" Command="Syncronize">
        <UCPTname>Net</UCPTname>
        <UCPTannotation>;action="SyncAll";clearHiddenFbs="true";sync="restart"</UCPTannotation>
    </Item>
  </iLonItem>
</InvokeCmd>
```

14.1.3.2 Issuing Network Scan Commands to Discover Devices

You can use the *InvokeCmd* function to discover all the uncommissioned devices on a LONWORKS network. When you send this command, a message is broadcast to the devices on the network that triggers the devices to identify themselves by their Neuron IDs.

The network scan automatically generates a Data Log named "<network>/**#DeviceDiscovery**" that includes entries for each uncommissioned device discovered by the network scan. Each entry includes a **Net/VirtCh/iLON System/VirtFb/LonDiscoveryMessage** data point that is set to the program ID and Neuron ID of a discovered device. You can periodically read the data log for uncommissioned devices while the network scan is ongoing based on the log's <UCPTlastUpdate> property, or you can wait until the network scan has been completed to read the uncommissioned devices in the data log all at once. See *Processing Discovered Devices* later in this section for more information.

Note: Section 21.1.6, *Discovering and Installing External Devices in Visual C# .NET*, includes a C# programming example demonstrating how to discover and install uncommissioned LONWORKS devices. Section 21.2.6, *Discovering and Installing External Devices in Visual Basic.NET*, describes how to do this in Visual Basic. *Section 22.3.4*, *Discovering and Installing External Devices in JAVA*, describes how to do this in JAVA.

The following example demonstrates how to scan a LONWORKS network for uncommissioned devices:

```
<InvokeCmd xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <Item xsi:type="LON_Network_ScanCommand_Invoke" ScanCommand="SetScan">
            <UCPTname>Net</UCPTname>
            <Command>
                <UCPTcommand>ScanOnce</UCPTcommand>
                <UCPTstatus xsi:type="string" LonFormat="UCPTstatus">STATUS_REQUEST</UCPTstatus>
            </Command>
            <UCPTscan LonFormat="UCPTscan">NST_ILON_DOMAIN</UCPTscan>
        </Item>
    </iLonItem>
</InvokeCmd>
```

The input parameters you supply to this function include one <Item> element with a LON_Network_Scan_Command_Invoke xsi type and a LONNetworkEScanCommand attribute that is set to "SetScan". The <Item> element must include the <UCPTname> of the network being scanned; a <UCPTcommand> that is set to "ScanOnce" to scan the network once; and a <UCPTscan> property that is set to "NST_ILON_DOMAIN". The following table lists the required properties of the <Item> element.

Option	Description
`<UCPTname>`	The name of the LONWORKS to be scanned for uncommissioned devices.
`<Command>`	A LONNetworkEScanCommand type that requires the following two properties: <UCPTcommand>. The network scan command. Set this property to **ScanOnce**, so that the SmartServer scans the network once. • <UCPTstatus>. An E_LonString type that indicates the status of the network scan command. When issuing a network scan command, you must set this property to **STATUS_REQUEST**. This property may be one of the following values: **STATUS_REQUEST**

Option	Description
	STATUS_CANCEL **STATUS_PENDING** **STATUS_DONE** **STATUS_FAIL** **STATUS_INVOKE** When this property is **STATUS_PENDING**, the network scan is ongoing. When this property is **STATUS_DONE**, the network scan is finished. See *Checking the Network Scan Status* for more information.
<UCPTscan>	An E_LonString type with a LonFormat attribute that must be set to <"UCPTscan">. This property must be set to **NST_ILON_DOMAIN**.

14.1.3.2.1 Checking the Network Scan Status

After you initiate the network scan, you can check its status to see whether it is still being performed or it has been completed. The following example demonstrates how to check the status of a network scan:

```
<iLonItem>
    <Item xsi:type="LON_Network_ScanCommand_Invoke" ScanCommand="GetScan">
        <UCPTname>Net</UCPTname>
    </Item>
</iLonItem>
```

To check the network scan status, you supply the *InvokeCmd* function one <Item> element with a LON_Network_Scan_Command_Invoke xsi type and a LONNetworkEScanCommand attribute that is set to "GetScan". The <Item> element only requires the <UCPTname> of the network being scanned.

The *InvokeCmd* function returns the status of the network scan.

```
<iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="LON_Network_ScanCommand_Invoke" ScanCommand="GetScan">
        <UCPTname>Net</UCPTname>
        <Command>
            <UCPTcommand>ScanOnce</UCPTcommand>
            <UCPTlastUpdate>2009-12-09T17:52:19.570-08:00</UCPTlastUpdate>
            <UCPTstatus LonFormat="UCPTstatus">STATUS_DONE</UCPTstatus>
        </Command>
        <UCPTscan>NST_ILON_DOMAIN</UCPTscan>
        <UCPTunCfgOnly>0</UCPTunCfgOnly>
    </Item>
</iLonItem>
```

14.1.3.2.2 Processing Discovered Uncommissioned Devices

When you scan a network for uncommissioned devices, the SmartServer automatically generates a Data Log named "<network>/**#DeviceDiscovery**" that includes entries for each uncommissioned device discovered by the network scan. Each entry includes a **Net/VirtCh/iLON System/VirtFb/LonDiscoveryMessage** data point that contains the program ID and Neuron ID of a discovered device. You will need to parse the program ID and Neuron ID from each data point, get the device template for each device based off the program ID, and then create and install the devices. The C#, VB, and JAVA programming example for discovering and installing uncommissioned devices in Chapters 21 and 22 demonstrate how to do this.

You can periodically read the data log for uncommissioned devices while the network scan is ongoing. This approach lets you begin processing uncommissioned devices without waiting; however, it is more complex. This is because you need to process the <UCPTtimeStamp> in the HEADER of the last

Read response and store it in the <UCPTlastUpdate> property of the subsequent *Read* request to avoid missing entries or receiving duplicate entries. This is the approach used by the SmartServer Web interface.

Alternatively, you can wait until the network scan has been completed to read the uncommissioned devices in the data log all at once. This approach requires you to wait for the scan to be completed, but is much simpler to program. The C#, VB, and JAVA programming example for discovering and installing uncommissioned devices in Chapters 21 and 22 use this approach.

14.1.4 Using the Delete Function on a LONWORKS Network

You can use the *Delete* function to delete a LONWORKS network in an LNS network database via the LNS Proxy Web service. You cannot delete a network via the SmartServer Web service as the current network is registered to the LON driver. To delete a LONWORKS network in an LNS network database, you provide the database's <UCPTname> property in the *Delete* Request as demonstrated in the following code sample:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>MyOldNetwork</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname> MyOldNetwork</UCPTname>
    </Item>
  </iLonItem>
```

14.2 LONWORKS Channels

The following section describes how to use the *List*, *Get*, *Set*, and *Delete* functions on LONWORKS channels.

14.2.1 Using the List Function on a LONWORKS Channel

You can use the *List* function to retrieve a list of LONWORKS channels on the SmartServer or to retrieve a list of LONWORKS channels in a specific LNS network database via the LNS Proxy Web service. The List function takes an <iLonItem> element that has an xSelect statement with a LON_Channel_Cfg type as its input, as shown in the example below.

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="LON_Channel_Cfg"]</xSelect>
  </iLonItem>
</List>
```

Response

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
    <Item>
        <UCPTname>Net/LON</UCPTname>
        <UCPTannotation>TP;xsi:type="LON_Channel_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
```

```
    <Item>
        <UCPTname>Net/LON IP</UCPTname>
        <UCPTannotation>IP;xsi:type="LON_Channel_Cfg"</UCPTannotation>
        <UCPThidden>1</UCPThidden>
    </Item>
  </iLonItem>
</List>
```

The *List* function returns a list of <Item> elements for each LONWORKS channel defined on the SmartServer or each LONWORKS channel defined in a specific LNS network database.

You could use the list of <Item> elements returned by this function as input for the *Get* function. The *Get* function would then return the configuration of each channel included in the list. The next section describes the properties included in each of these elements.

14.2.2 Using the Get Function on a LONWORKS Channel

You can use the *Get* function to retrieve the configuration of a LONWORKS channel defined on the SmartServer or a LONWORKS channel in a specific LNS network database. The input parameters you supply to this function will include one or more <Item> elements with a LON_Channel_Cfg type. Each <Item> element will include the <UCPTname> of each channel whose configuration is to be returned by this function, as shown in the example below.

Request

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Channel_Cfg">
        <UCPTname>Net/LON</UCPTname>
    </Item>
  </iLonItem>
</Get>
```

Response

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="LON_Channel_Cfg">
        <UCPTname>Net/LON</UCPTname>
        <UCPTannotation>TP;xsi:type="LON_Channel_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-20T17:42:54.903-07:00</UCPTlastUpdate>
        <UCPTuri>LON_Channel_Cfg.htm</UCPTuri>
        <UCPThandle>0</UCPThandle>
        <UCPTchannelType LonFormat="UCPTchannelType">CT_LON</UCPTchannelType>
        <UCPTtransceiverId LonFormat="UCPTtransceiverId">TP_FT_10</UCPTtransceiverId>
        <UCPTtransmitTimer LonFormat="UCPTtransmitTimer">TT_96</UCPTtransmitTimer>
        <UCPTretryCount>3</UCPTretryCount>
        <UCPTmaxPriority>4</UCPTmaxPriority>
        <UCPTchannelMode LonFormat="UCPTchannelMode">AM_ONLINE</UCPTchannelMode>
        <UCPTdynamic LonFormat="UCPTdynamic">DDT_STATIC</UCPTdynamic>
    </Item>
  </iLonItem>
</Get>
```

The *Get* function returns an <Item> element for each channel referenced in the input parameters you supplied to the function. The properties included within each <Item> element are initially defined when the channel is added to the SmartServer or LNS network database. You can write to these channel properties with the *Set* function. The following table describes these properties.

Property	Description
<UCPTname>	The name of the channel in the following format: <*network*/*channel*>. You can rename a LONWORKS channel by providing its <UCPThandle> and specifying the new

Property	Description
	<UCPTname> property to which the channel is to be renamed.
<UCPTannotation>	The type of channel and its xsi type, which is LON_Channel_Cfg. This determines the icon used to represent the channel in the SmartServer Web interface.
<UCPThidden>	A flag indicating whether the channel is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **0** – shown **1** – hidden
<UCPTlastUpdate>	A timestamp indicating the last time the configuration of the channel was updated. This timestamp uses the ISO 8601 format, which is as follows: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Data Point was last updated. The second segment (after the **T):** HH:MM:SS.sss represents the time of day the configuration of the Data Point 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 be expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. It can be left out. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002. If it is not UTC, time shift has to be defined: 2008-02-28T09:59:53.660+01:00
<UCPTdescription>	A user-defined description of the channel. This can be a maximum of 201 characters long.
<UCPTuri>	The name of the file on the SmartServer flash disk containing the configuration of the LONWORKS channel. This property is always LON_Channel_Cfg.htm.
<UCPThandle>	The handle of the channel assigned by the LNS server. When you use the *Set* function to modify the configuration of an existing channel, you must specify the channel's handle. If you do not specify the handle, a new channel is created. You cannot use the *Set* function to modify the handle assigned to the channel.
<UCPTchannelType>	The channel type. This value is typically CT_LON. This value must be set to CT_LON_REPEATING if the channel

Property	Description
	is a power line channel that uses the Enhanced LonTalk® Proxy Protocol to transmit network messages from the SmartServer to the devices attached to the channel. For more information on the Enhanced LonTalk Proxy Protocol and power line repeating, see the *i.LON SmartServer Power Line Repeating Network Management Guide*.
`<UCPTtransceiverId>`	The transceiver ID of the channel. To create a new channel with the *Set* function, you must include this property. This property may be one of the following values: IP_10L IP_10W TP_FT_10 TP_XF_78 TP_XF_1250 PL_20C PL_20N PL_20C_LOW PL_20A PL_20A_LOW TP_RS485_39 TP_RS485_78 TP_RS485_625 TP_RS485_1250 RF_10 CUSTOM
`<UCPTtransmitTimer>`	The interval (in milliseconds) network messages wait for confirmation before being re-sent over the network. The default value is TT_96 for FT-10 channels and TT_512 for PL channels. The default interval is calculated based on the network topology, specifically the transmission time for each channel that the message must cross. By default, the transmission time for each channel is determined by its type.
`<UCPTretryCount>`	The number of times a network message is re-sent when no confirmation is received. The default value is 3 for FT-10 channels, and it is 5 for PL-20 channels. The default retry count, which can range from 0 to 15 attempts, is calculated based on network topology. Typically, the default retry count is set to 3 attempts; however, if a message must pass through certain channel types, the default may be increased. For example, if a message must cross a PL-20 channel, the default retry count would be increased to 5 attempts.
`<UCPTmaxPriority>`	The maximum number of priority slots available on the channel. Priority slots may be used by the critical devices on a network that use priority messaging. With priority messaging, the device with the highest priority sends its packet before any other devices can send theirs. This

Property	Description
	is accomplished by assigning each priority device a time (priority) slot where it can transmit before all other lower priority and non-priority devices. These time slots consume network bandwidth; therefore, priority messaging should only be used for critical devices and data.
<UCPTdelay>	The expected longest round-trip time (in milliseconds) of a message (for example, message and response). This option allows expected traffic patterns to be input into the system so that the timer calculations can be affected accordingly. The default round-trip delay is two packet cycles based on the average packet size. This property is optional. If you do not specify this property in a *Set* function, the current value stored in it is erased. You must specify this property even if you are not changing it in order to preserve the current value.
<UCPTchannelMode>	This property is always set to AM_ONLINE.
<UCPTminOfflineTime>	If a network message fails, a data point and its device are marked offline. You can specify the <UCPTminOfflineTime> property so that all the data points on the offline device with pending network messages (read/write requests, polls, or heartbeats) are marked offline and network messages are not sent to them. This ensures that network performance is not impacted by an offline device. You can set the minimum period of time (in seconds) that the SmartServer waits before transmitting network messages to offline data points. During this period, an offline device transmits an OFFLINE status in response to data point requests. Once <UCPTminOfflineTime> elapses, the SmartServer sends a read/write request to one offline data point. If the read/write request succeeds, the data point and its device are marked online, and all cached read/write requests for the offline data points on the device are executed. This property is optional. If you do not specify this property in a *Set* function, the current value stored in it is erased. You must specify this property even if you are not changing it in order to preserve the current value.
<UCPTofflineIhbD>	You can set the period of time (in seconds) that a device executes data point requests from the SmartServer after one request fails. Once this timer expires, the device transmits an OFFLINE status in response to data point requests. This property is optional. If you do not specify this property in a *Set* function, the current value stored in it is erased. You must specify this property even if you are not changing it in order to preserve the current value.
<UCPTdynamic>	Indicates whether the channel is static (DDT_STATIC) or dynamic (DDT_DYNAMIC). All channels in the LNS Proxy

Property	Description
	Web service should be set to DDT_DYNAMIC. You cannot use the *Set* function to modify this property

14.2.3 Using the Set Function on a LONWORKS Channel

You can use the *Set* function to overwrite the configuration of a channel, or to create a new channel. The input parameters you supply to the function will include one or more <Item> elements. Each <Item> element includes a <UCPTname> property that specifies a unique channel to be created or modified.

Each <Item> element may also include a series of properties that define the configuration of the new (or modified) channel. This set of properties is the same whether you are creating a new channel or modifying an existing channel.

- If you are creating a new channel, you only need to specify the <UCPTtransceiverId> property; all other properties are optional.
- If you are modifying an existing channel, you must specify the channel's <UCPThandle>. If you do not specify the handle, a new channel is created. All other properties must be filled; otherwise the values stored in them are erased. The previous section, *Using the Get Function on a LONWORKS Channel*, details the properties you can include in the *Set* function.

You can set multiple channels with a single *Set* message. However, you should not attempt to create or write to more than 100 channels with a single call to the *Set* function.

Request (create a new LONWORKS channel on the SmartServer)

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Channel_Cfg">
        <UCPTname>MyNewNetwork/myNewChannel</UCPTname>
        <UCPTannotation>TP</UCPTannotation>
        <UCPTtransceiverId LonFormat="UCPTtransceiverId">TP_FT_10</UCPTtransceiverId>
    </Item>
  </iLonItem>
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>MyNewNetwork/myNewChannel</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

14.2.4 Using the Delete Function on a LONWORKS Channel

You can use the *Delete* function to delete a LONWORKS channel on the SmartServer or a channel in an LNS network database via the LNS Proxy Web service. The *Delete* function takes an <Item> element with a LON_Channel_Cfg type as its input. The <Item> element only needs to include the channel's <UCPTname> property in the *Delete* Request as demonstrated in the following code sample:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Channel_Cfg">
        <UCPTname>MyOldNetwork/MyOldChannel</UCPTname>
    </Item>
  </iLonItem>
```

```
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>MyOldNetwork/MyOldChannel</UCPTname>
    </Item>
  </iLonItem>
```

14.3 LONWORKS Devices

The following section describes how to use the *List*, *Get*, *Set*, and *Delete* functions on LONWORKS devices.

Note: Section 21.1.4, *Creating and Installing a LonWorks Device in Visual Basic.NET*, includes a C# programming example demonstrating how to use the LONWORKS Device SOAP interface to create and install LONWORKS devices. Section 21.2.4, *Creating and Installing a LonWorks Device in Visual Basic.NET*, includes a Visual Basic example demonstrating how to do this.

Section 22.3.3, *Creating and Installing a LonWorks Device in Java*, includes a Java programming example demonstrating how to create and install external LONWORKS devices.

14.3.1 Using the List Function on a LONWORKS Device

You can use the *List* function to retrieve a list of LONWORKS devices on the SmartServer or to retrieve a list of LONWORKS devices in a specific LNS network database via the LNS Proxy Web service. The List function takes an <iLonItem> element that has an xSelect statement with a LON_Device_Cfg type as its input, as shown in the example below.

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="LON_Device_Cfg"]</xSelect>
  </iLonItem>
</List>
```

Alternatively, you can specify one or more device properties in the xSelect statement to filter the items returned by the *List* function, including the <UCPTname>, <UCPTitemStatus>, and <UCPTlastUpdate> properties.

Request (return all devices on a specific LONWORKS channel)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect> //Item[@xsi:type="LON_Device_Cfg"][starts-with(UCPTname,"MyNetwork/Channel 1")]
    </xSelect>
  </iLonItem>
</List>
```

Request (return all devices that are uncommissioned)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect> //Item[@xsi:type="LON_Device_Cfg"] [UCPTitemStatus="IS_UNCONFIGURED"]
    </xSelect>
  </iLonItem>
</List>
```

Request (return all devices that were updated after a specific time)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect> //Item[@xsi:type="LON_Device_Cfg"] [UCPTlastUpdate&gt; "2008-03-26T16:25:00"]
    </xSelect>
  </iLonItem>
</List>
```

Response

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
    <Item>
        <UCPTname>Net/LON/iLON App</UCPTname>
        <UCPTannotation>900001012881040c;xsi:type="LON_Device_Cfg";local</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
    <Item>
        <UCPTname>Net/LON/AI-1</UCPTname>
        <UCPTannotation>App;xsi:type="LON_Device_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
    <Item>
        <UCPTname>Net/LON/AO-1</UCPTname>
        <UCPTannotation>App;xsi:type="LON_Device_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
    <Item>
        <UCPTname>Net/LON/DIO-1</UCPTname>
        <UCPTannotation>App;xsi:type="LON_Device_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
    <Item>
        <UCPTname>Net/LON/DIO-2</UCPTname>
        <UCPTannotation>App;xsi:type="LON_Device_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
  </iLonItem>
</List>
```

You could use the list of <Item> elements returned by this function as input for the *Get* function. The *Get* function would then return the configuration of each device included in the list. The next section describes the properties included in each of these elements.

14.3.2 Using the Get Function on a LONWORKS Device

You can use the *Get* function to retrieve the configuration of a LONWORKS device defined on the SmartServer or in a specific LNS network database. The input parameters you supply to this function will include one or more <Item> elements with a LON_Device_Cfg type. Each <Item> element will include the <UCPTname> of each device whose configuration is to be returned by this function, as shown in the example below.

Alternatively, you can specify one or more device properties in the xSelect statement to filter the items returned by the *Get* function, including the <UCPTname> to filter devices based on their parent channel; <UCPThidden> to filter devices based on whether they are hidden or shown in the SmartServer Web interface; <UCPTlocal> to filter devices based on whether they are external or internal to the SmartServer; <UCPTitemStatus> to filter devices based on whether they are uncommissioned, offline, or out of sync with an LNS network database; and <UCPTprogramID> to filter devices based on the manufacturer.

Request (return a specific device)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Device_Cfg">
        <UCPTname>Net/LON/DIO-1</UCPTname>
    </Item>
  </iLonItem>
</Get>
```

Request (use an xSelect statement return all the devices on a specific LONWORKS channel)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect> //Item[@xsi:type="LON_Device_Cfg"][starts-with(UCPTname,"Net/LON")]</xSelect>
  </iLonItem>
</Get>
```

Request (use an xSelect statement return all internal devices on the SmartServer)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>
      //Item[@xsi:type="LON_Device_Cfg"] [UCPTlocal="1"]
      </xSelect>
  </iLonItem>
</Get>
```

Request (use an xSelect statement return all external devices that are currently shown in the SmartServer Web interface)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>
      //Item[@xsi:type="LON_Device_Cfg"][UCPTlocal="0"][UCPThidden="0"]
      </xSelect>
  </iLonItem>
</Get>
```

Request (use an xSelect statement return any devices that are out of sync with an LNS network database)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>
      //Item[@xsi:type="LON_Device_Cfg"] [UCPTitemStatus="IS_NOTSYNCED”]
      </xSelect>
  </iLonItem>
</Get>
```

Request (use an xSelect statement return any devices that are uncommissioned)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
```

```
    <xSelect>
      //Item[@xsi:type="LON_Device_Cfg"] [UCPTitemStatus="IS_UNCONFIGURED”]
      </xSelect>
  </iLonItem>
</Get>
```

Request (use an xSelect statement return any devices that are offline)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>
      //Item[@xsi:type="LON_Device_Cfg"] [UCPTitemStatus="IS_APP_STOPPED”]
      </xSelect>
  </iLonItem>
</Get>
```

Request (use an xSelect statement to return all devices from a specific manufacturer using the program ID)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>
      //Item[@xsi:type="LON_Device_Cfg"] [UCPTprogramId="80000105288a0403"] </xSelect>
  </iLonItem>
</Get>
```

Response

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Device_Cfg">
        <UCPTname>Net/LON/DIO-1</UCPTname>
        <UCPTannotation>App;xsi:type="LON_Device_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-26T17:09:46.160-07:00</UCPTlastUpdate>
        <UCPTuri>LON_Device_Cfg.htm</UCPTuri>
        <UCPThandle>5</UCPThandle>
        <UCPTuniqueId>00a145791500</UCPTuniqueId>
        <UCPTprogramId>80000105288a0403</UCPTprogramId>
        <UCPTgeoPosition>1</UCPTgeoPosition>
        <UCPTlocationId>000000000000</UCPTlocationId>
        <UCPTmaxDynamicFb>0</UCPTmaxDynamicFb>
        <UCPTmaxDynamicDp>0</UCPTmaxDynamicDp>
        <UCPTmaxTxTransactions>0</UCPTmaxTxTransactions>
        <UCPTmaxTxLifetime>0</UCPTmaxTxLifetime>
        <UCPTlocal>0</UCPTlocal>
        <UCPTapplicationStatus LonFormat="UCPTapplicationStatus">APP_RUNNING</UCPTapplicationStatus>
        <UCPTcommissionStatus LonFormat="UCPTcommissionStatus">COMMISSIONED</UCPTcommissionStatus>
        <UCPTurlImage />
        <UCPTurlTemplate>/root/lonWorks/Import/Echelon/LonPoint/Version3/dio-10v3.xif</UCPTurlTemplat
        e>
        <UCPTdynamic LonFormat="UCPTdynamic">DDT_DYNAMIC</UCPTdynamic>
        <Address>
            <UCPTdomainIndex>0</UCPTdomainIndex>
            <UCPTdomainLength>6</UCPTdomainLength>
            <UCPTdomainKey>7DEF1857C766</UCPTdomainKey>
            <UCPTsubnet>1</UCPTsubnet>
            <UCPTnodeId>5</UCPTnodeId>
        </Address>
        <Address>
            <UCPTdomainIndex>1</UCPTdomainIndex>
            <UCPTdomainLength>0</UCPTdomainLength>
            <UCPTdomainKey />
            <UCPTsubnet>0</UCPTsubnet>
            <UCPTnodeId>0</UCPTnodeId>
        </Address>
        <Command>
            <UCPTcommand>Wink</UCPTcommand>
            <UCPTlastUpdate>2008-03-26T17:09:46.160-07:00</UCPTlastUpdate>
            <UCPTstatus LonFormat="UCPTstatus">STATUS_DONE</UCPTstatus>
        </Command>
    </Item>
```

```
  </iLonItem>
</Get>
```

The *Get* function returns an <Item> element for each device referenced in the input parameters you supplied to the function. The properties included within each <Item> element are initially defined when the device is added to the SmartServer or LNS network database. You can write to these device properties with the *Set* function. The following table describes these properties.

Property	Description
`<UCPTname>`	The name of the device in the following format: <*network*/*channel*/*device*>. You can rename a LONWORKS device by providing its <UCPThandle> and specifying the new <UCPTname> property to which the device is to be renamed.
`<UCPTannotation>`	The type of device (**App** by default) and its xsi type, which is LON_Device_Cfg. This determines the icon used to represent the device in the SmartServer Web interface. You can use custom icons (.gif images) to represent your company's devices in the SmartServer Web interface. If a custom icon is being used for a device, this property is set to *programID*. To use a custom device icon, upload the *programID***.gif** file for your custom device icon to both the root/Web/images/tree and the root/Web/images/app folders on the SmartServer flash disk. The root/web/images/tree stores the icons shown in the SmartServer tree. The root/web/images/app folder stores the icons shown in the upper left-hand corner of an object's configuration and driver Web pages.
`<UCPThidden>`	A flag indicating whether the device is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **`0`** – shown **`1`** – hidden
`<UCPTitemStatus>`	This property only appears if the device has one of the following exceptions: • IS_UNCONFIGURED. The device is uncommissioned. • IS_APP_STOPPED. The device application is offline. • IS_DELETED. The device has been deleted. • IS_NOTSYNCED. The device is out of sync with the LNS network database.
`<UCPTlastUpdate>`	A timestamp indicating the last time the configuration of the device was updated. This timestamp uses the ISO 8601 format, which is as follows: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Data Point was last updated. The second segment (after the **T):** HH:MM:SS.sss

Property	Description
	represents the time of day the configuration of the Data Point 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 be expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. It can be left out. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002. If it is not UTC, time shift has to be defined: 2008-02-28T09:59:53.660+01:00
`<UCPTdescription>`	A user-defined description of the device. This can be a maximum of 201 characters long.
`<UCPTuri>`	The name of the file on the SmartServer flash disk containing the configuration of the LONWORKS device. This property is always LON_Device_Cfg.htm.
`<UCPThandle>`	The handle of the device assigned by the LNS server. When you use the *Set* function to modify the configuration of an existing device, you must specify the device's handle. If you do not specify the handle, a new device is created. You cannot use the *Set* function to modify the handle assigned to the device.
`<UCPTuniqueId>`	The current Neuron ID of the device. The Neuron ID is a unique 48-bit number that is burnt into the Neuron chip of an external device or assigned to one of the SmartServer's 16 internal devices.
`<UCPTprogramId>`	The program ID of the device as a set of hex digits. The program ID uniquely defines the static portion of the device interface. It is hard coded into an external device (in the EEPROM of the device's Neuron Chip), and it is assigned the 16 internal devices on the SmartServer. You should not change this property for external or internal devices.
`<UCPTgeoPosition>`	The waypoint of the device. A waypoint is a set of coordinates (latitude and longitude) that identifies the device's location in physical space. Typically, waypoints are acquired with a GPS and then uploaded to the SmartServer using SOAP/HTTP messages over the console port.
`<UCPTlocationId>`	The 6-byte hexadecimal location string that documents the device's location within the network.
`<UCPTmaxDynamicFb>`	The maximum number of dynamic functional blocks that you can add to the device. A dynamic functional block is a functional block that is not

Property	Description
	pre-loaded on a device. Devices that support dynamic functional blocks include controllers that do not have a static interface. For example, the v40 SmartServer XIF, which has a dynamic interface, supports a maximum of 500 dynamic functional blocks.
<UCPTmaxDynamicDp>	The maximum number of dynamic network variables/data points that you can add to the device. A dynamic network variable/data point can be added to a functional block after the device has been commissioned. Devices that support dynamic network variables/data points include controllers and gateways with dynamic interfaces. For example, the v40 SmartServer XIF, which has a dynamic interface, supports a maximum of 3000 dynamic network variables/data points.
<UCPTmaxTxTransactions>	The maximum number of simultaneous transactions supported by the device application. If the device application exceeds this maximum value, then any attempt to begin a new transaction will fail.
<UCPTmaxTxLifetime>	Displays the timeout value (in milliseconds) for a transaction. This value represents the longest period of time a transaction can be active.
<UCPTlocal>	A flag indicating whether the device is internal or external to the SmartServer. This property may be one of the following values: **0** – External. An application device that is physically installed on the network. **1** – Internal. An emulation of a device that resides on the SmartServer and encapsulates the functional blocks and data points within an XIF or template. The SmartServer contains 16 internal devices. One of these internal devices is the SmartServer automated systems device (the **iLON App** device), which contains the SmartServer's built-in embedded applications. Ten of the internal devices are reserved for the custom embedded applications (called Freely Programmable Modules [FPMs]) that you can write and deploy on your SmartServer using the full version of i.LON FPM Programming tools. The other five internal devices on the SmartServer consist of the **iLON System** device in which all the virtual data points (formerly referred to as NVVs) are stored, the IP-852 router, the local network interface, the RNI, and the LonTalk device.
<UCPTapplicationStatus>	Indicates the current device configuration, which can be one of the following values: • AP_NUL. Never reached.

Property	Description
	• APP_RUNNING. Online. • APP_STOPPED. Offline.
<UCPTcommissionStatus>	Indicates the current device configuration, which can be one of the following values: • COMSTATE_NUL. Never reached. • COMMISSIONED. Configured. • UNCOMMISSIONED. Unconfigured.
<UCPTurlImage>	The full path on the SmartServer flash disk of an application image to be downloaded to the device by the SmartServer. You can use the SmartServer to upgrade a Neuron-hosted device that has writeable application memory (EEPROM or flash). An upgrade may be needed to improve the device's capabilities or to repair a damaged device application.
<UCPTurlTemplate>	The full path on the SmartServer flash disk of the external interface (.XIF or .XML file) loaded on the SmartServer for the device. The external interface is the logical interface to a device. A device's external interface specifies the number and types of functional blocks, and the number, types, directions, and connection attributes of data points. The program ID field is used as the key to identify each external interface. Each program ID uniquely defines the static portion of the interface. However, two devices with identical static portions may differ if dynamic data points are added or removed, or if the types of changeable data points are modified. Thus it is possible to have devices with the same program ID but different external interfaces.
<UCPTdynamic>	Indicates whether the device is static (DDT_STATIC) or dynamic (DDT_DYNAMIC). All devices in the LNS Proxy Web service should be set to DDT_DYNAMIC. You cannot use the *Set* function to modify this property
<Address>	Each device contains a set of <Address> elements listing the domainKey, subnet, and node of their primary and secondary addresses. A device may have a secondary address if it is a member of another network. The domainKey is the domain ID of the network. The subnet/node ID is used for addressing messages. The subnet ID identifies the channel (subnet) on which the device resides, and the node ID identifies the device on that channel. The subnet/node IDs begin with an address of 1/1 and increase sequentially to 1/2, 1/3, and so on for devices on the same channel (subnet). For a second channel created on the network, the subnet/node IDs would begin with an address of 2/1 and increase sequentially to 2/2, 2/3, and so on. This property contains the following child elements

Property	Description
<Command>	Lists the network management and debugging commands issued for the device and their statuses. You can issue network management commands (ChangeApplicationStatus, ChangeCommissionStatus, ImageDownload, GetTemplate, Wink, and FetchProgID) in a *Set* function. You can issue debugging commands (QueryStatus and ClearStatus) with the *Invoke_Cmd* function. Note that these commands require the Item type of LON_Device_Command_Invoke. To issue a network management or debugging command, you provide the desired command, any required properties as described in this table, and setting the <UCPTstatus> property to STATUS_REQUEST. Each <Command> object contains the following properties: • <UCPTcommand>. The network management or debugging command issued. See *Issuing Network Management Commands* and *Issuing Debugging Commands* for more information. • <UCPTlastUpdate time>. Indicates when the status of the command was last changed. • <UCPTstatus>. The status of the network management command. When issuing a network management or debugging command, you must set this property to STATUS_REQUEST. This property may be one of the following values: • STATUS_REQUEST • STATUS_CANCEL • STATUS_PENDING • STATUS_DONE • STATUS_FAIL • STATUS_INVOKE

14.3.3 Using the Set Function on a LONWORKS Device

You can use the *Set* function to overwrite the configuration of a device, or to create a new device. The input parameters you supply to the function will include one or more <Item> elements. Each <Item> element includes a <UCPTname> property that specifies a unique device to be created or modified.

Each <Item> element may also include a series of properties that define the configuration of the new (or modified) device. This set of properties is the same whether you are creating a new device or modifying an existing device.

- If you are creating a new device, you only need to specify the <UCPTtransceiverId> property; all other properties are optional. You can commission the device and set the device application online by providing the <UCPTcommissionStatus> and <UCPTapplicationStatus> properties and setting them to COMMISSIONED and APP_RUNNING respectively. Otherwise, the device will be decommissioned and/or set offline upon creation.
- If you are modifying an existing device, you must specify the <UCPThandle> property. In addition, all other properties should be filled; otherwise the values stored in them are erased. The previous section, *Using the Get Function on a LONWORKS Device*, details the properties you can include in the *Set* function.

You can set multiple devices with a single *Set* message. However, you should not attempt to create or write to more than 100 devices with a single call to the *Set* function. The following example demonstrates how to create a new LONWORKS device. Note that in this example, the device interface (XIF) file is also activated. This automatically creates the device's functional blocks and data points without having to explicitly add them using a *Set* command. See *Issuing Network Management Commands* for more information on using the GetTemplate network management command.

Request (create a new LONWORKS device on the SmartServer and activate the device interface)

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
 <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="LON_Device_Cfg">
        <UCPTname>Net/LON/DIO-2</UCPTname>
        <UCPTuniqueId>00a145784600</UCPTuniqueId>
        <UCPTprogramId>80000105288a0403</UCPTprogramId>
        <UCPTapplicationStatus LonFormat="UCPTapplicationStatus">APP_RUNNING</UCPTapplicationStatus>
        <UCPTcommissionStatus LonFormat="UCPTcommissionStatus">COMMISSIONED</UCPTcommissionStatus>
        <UCPTurlTemplate>/root/lonWorks/Import/Echelon/LonPoint/Version3/dio-10v3.xif</UCPTurlTemplat
        e>
        <Command>
            <UCPTcommand>GetTemplate</UCPTcommand>
            <UCPTstatus LonFormat="UCPTstatus">STATUS_REQUEST</UCPTstatus>
        </Command>
    </Item>
  </iLonItem>
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/LON/DIO-2</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

14.3.3.1 Issuing Network Management Commands

You can use the *Set* function to issue network management commands on LONWORKS devices. You can use network management commands to commission/decommission devices, set device applications online/offline, upgrade devices by downloading application images files (.apb extension) to them, activate device templates, reset devices, wink devices, and fetch the program IDs of devices.

To issue a network management command, you need to provide one or more <Item> elements with a LON_Device_Cfg type. Each <Item> element needs to include the <UCPTname>, <UCPThandle>, <UCPTuniqueId>, and <UCPTprogramId> properties of the device upon which network management commands are to be issued. In addition, you should provide any other properties whose values are to be preserved (if you do not provide a property, the current value stored in it is erased). Particularly, you should provide the <UCPTcommissionStatus and <UCPTapplicationStatus> properties when appropriate; otherwise, the device may be decommissioned and set offline. Specific network management commands require additional properties as described in the following sections that describe how to use each command.

Note: Section 21.1.5, *Commissioning External Devices in Visual Basic.NET*, includes a C# programming example demonstrating how to issue network management commands in the LONWORKS Device SOAP interface in order to commission unconfigured external LONWORKS devices. Section 21.2.5, *Commissioning External Devices in Visual Basic.NET*, includes a Visual Basic example demonstrating how to do this.

ChangeApplicationStatus

You can use this command to set the device application online or offline. This command requires the desired <UCPTapplicationStatus> property, which may be APP_RUNNING (online) or APP_STOPPED (offline).

If you are setting a device application offline, you should still provide the <UCPTcommissionStatus> property and set it to COMMISSIONED; otherwise, the commission status will be changed to COMSTATE_NUL (never reached) and the device is decommissioned. The following example demonstrates how to set a device application offline.

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Device_Cfg">
        <UCPTname>Net/LON/DIO-1</UCPTname>
        <UCPThandle>5</UCPThandle>
        <UCPTuniqueId>00a145791500</UCPTuniqueId>
        <UCPTprogramId>80000105288a0403</UCPTprogramId>
        <UCPTcommissionStatus LonFormat="UCPTcommissionStatus">COMMISSIONED</UCPTcommissionStatus>
        <UCPTapplicationStatus>APP_STOPPED</UCPTapplicationStatus>
        <Command>
            <UCPTcommand>ChangeApplicationStatus</UCPTcommand>
            <UCPTstatus LonFormat="UCPTstatus">STATUS_REQUEST</UCPTstatus>
        </Command>
    </Item>
  </iLonItem>
</Set>
```

ChangeCommissionStatus

You can use this command to commission or decommission a device. If you are commissioning a device, you should also set the device application online using the ChangeApplicationStatus command. If you are decommissioning a device, you need to provide the <UCPTcommissionStatus> property and set it to DECOMMISSIONED. The following example demonstrates how to commission a device and set the device application online.

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Device_Cfg">
        <UCPTname>Net/LON/DIO-1</UCPTname>
        <UCPThandle>5</UCPThandle>
        <UCPTuniqueId>00a145791500</UCPTuniqueId>
        <UCPTprogramId>80000105288a0403</UCPTprogramId>
        <UCPTapplicationStatus>APP_RUNNING</UCPTapplicationStatus>
        <UCPTcommissionStatus LonFormat="UCPTcommissionStatus">COMMISSIONED</UCPTcommissionStatus>
        <Command>
            <UCPTcommand>ChangeApplicationStatus</UCPTcommand>
            <UCPTstatus LonFormat="UCPTstatus">STATUS_REQUEST</UCPTstatus>
        </Command>
        <Command>
            <UCPTcommand>ChangeCommissionStatus</UCPTcommand>
```

```
            <UCPTstatus LonFormat="UCPTstatus">STATUS_REQUEST</UCPTstatus>
        </Command>
    </Item>
  </iLonItem>
</Set>
```

ImageDownload

You can use the ImageDownload command to download an application image file (.apb extension) stored on the SmartServer flash disk to the device. When using the ImageDownload command, you must specify the desired <UCPTurlImage> of the application image file to be downloaded to the device. If the device interface of the device being loaded has changed, you can include the <UCPTurlTemplate> property and set it to the full path of the external interface file (.XIF extension) or device template (.XML extension) on the SmartServer flash disk, and then use the GetTemplate command to activate the new device interface on the SmartServer. The following example demonstrates how to download an application to a device and activate a new device interface file on the SmartServer.

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Device_Cfg">
        <UCPTname>Net/LON/AI-1</UCPTname>
        <UCPThandle>3</UCPThandle>
        <UCPTuniqueId>000256654500</UCPTuniqueId>
        <UCPTprogramId>80000105188a0403</UCPTprogramId>
        <UCPTapplicationStatus LonFormat="UCPTapplicationStatus">APP_RUNNING</UCPTapplicationStatus>
        <UCPTcommissionStatus LonFormat="UCPTcommissionStatus">COMMISSIONED</UCPTcommissionStatus>
        <UCPTurlImage>/root/lonWorks/import/Echelon/LonPoint/Version3/ai-10v3.apb</UCPTurlImage>
        <UCPTurlTemplate>/root/lonWorks/Import/Echelon/LonPoint/Version3/ai-10v3.xif</UCPTurlTemplate
        >
        <Command>
            <UCPTcommand>ImageDownload</UCPTcommand>
            <UCPTstatus LonFormat="UCPTstatus">STATUS_REQUESTUCPTstatus>
        </Command>
        <Command>
            <UCPTcommand>GetTemplate</UCPTcommand>
            <UCPTstatus LonFormat="UCPTstatus">STATUS_REQUEST</UCPTstatus>
        </Command>
    </Item>
  </iLonItem>
</Set>
```

GetTemplate

You can use the GetTemplate command to activate a new device interface on the SmartServer. When using the GetTemplate command, you must specify the device's <UCPTurlTemplate> property and set it to the full path of the external interface file (.XIF extension) or device template (.XML extension) on the SmartServer flash disk. The following example demonstrates how to activate a new device interface file on the SmartServer.

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Device_Cfg">
        <UCPTname>Net/LON/AI-1</UCPTname>
        <UCPThandle>3</UCPThandle>
        <UCPTuniqueId>000256654500</UCPTuniqueId>
        <UCPTprogramId>80000105188a0403</UCPTprogramId>
        <UCPTapplicationStatus LonFormat="UCPTapplicationStatus">APP_RUNNING</UCPTapplicationStatus>
        <UCPTcommissionStatus LonFormat="UCPTcommissionStatus">COMMISSIONED</UCPTcommissionStatus>
        <UCPTurlTemplate>/root/lonWorks/Import/Echelon/LonPoint/Version3/ai-10v3.xif</UCPTurlTemplate
        >
        <Command>
            <UCPTcommand>GetTemplate</UCPTcommand>
            <UCPTstatus LonFormat="UCPTstatus">STATUS_REQUEST</UCPTstatus>
        </Command>
    </Item>
  </iLonItem>
</Set>
```

Reset

You can use the Reset command to stop a device application, terminate all incoming and outgoing messages, set all temporary settings to their initial values, and then restart the device application. The following example demonstrates how to reset a device.

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
   <iLonItem>
    <Item xsi:type="LON_Device_Cfg">
        <UCPTname>Net/LON/DIO-1</UCPTname>
        <UCPThandle>5</UCPThandle>
        <UCPTuniqueId>00a145791500</UCPTuniqueId>
        <UCPTprogramId>80000105288a0403</UCPTprogramId>
        <UCPTapplicationStatus LonFormat="UCPTapplicationStatus">APP_RUNNING</UCPTapplicationStatus>
        <UCPTcommissionStatus LonFormat="UCPTcommissionStatus">COMMISSIONED</UCPTcommissionStatus>
        <Command>
            <UCPTcommand>Reset</UCPTcommand>
            <UCPTstatus LonFormat="UCPTstatus">STATUS_REQUEST</UCPTstatus>
        </Command>
    </Item>
  </iLonItem>
</Set>
```

Wink

You can use the Wink command to identify a LONWORKS device on the network and verify that it is communicating properly. A device that supports the Wink command generates an application-dependent audio or visual feedback such as a beep or a flashing service LED when winked. When using the Wink command, you should provide the <UCPTcommissionStatus and <UCPTapplicationStatus> properties; otherwise, the device will be decommissioned and set offline. The following example demonstrates how to wink a device.

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
   <iLonItem>
    <Item xsi:type="LON_Device_Cfg">
        <UCPTname>Net/LON/DIO-1</UCPTname>
        <UCPThandle>5</UCPThandle>
        <UCPTuniqueId>00a145791500</UCPTuniqueId>
        <UCPTprogramId>80000105288a0403</UCPTprogramId>
        <UCPTapplicationStatus LonFormat="UCPTapplicationStatus">APP_RUNNING</UCPTapplicationStatus>
        <UCPTcommissionStatus LonFormat="UCPTcommissionStatus">COMMISSIONED</UCPTcommissionStatus>
        <Command>
            <UCPTcommand>Wink</UCPTcommand>
            <UCPTstatus LonFormat="UCPTstatus">STATUS_REQUEST</UCPTstatus>
        </Command>
    </Item>
  </iLonItem>
</Set>
```

FetchProgID

You can use the FetchProgID command to retrieves the <UCPTprogramId> property of the device. Note that this command is not required when a network is synchronized with an LNS network database. This is because the *Get* function automatically fetches the program ID, if needed. The following example demonstrates how to retrieve the program ID of a device:

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Device_Cfg">
        <UCPTname>Net/LON/DIO-1</UCPTname>
        <UCPThandle>5</UCPThandle>
        <UCPTuniqueId>00a145791500</UCPTuniqueId>
        <Command>
            <UCPTcommand>FetchProgID</UCPTcommand>
            <UCPTstatus LonFormat="UCPTstatus">STATUS_REQUEST</UCPTstatus>
        </Command>
    </Item>
  </iLonItem>
</Set>
```

14.3.3.2 Issuing Debugging Commands

You can use the *InvokeCmd* function to issue debugging commands on LONWORKS devices. The debugging commands consist of QueryStatus, ClearStatus, and SendServicePin. You can use the QueryStatus debugging command to test the performance of a device and diagnose any problems. You can use the ClearStatus debugging command to clear the device statistics returned by the QueryStatus command. You can use the SendServicePin command to send a service pin message from one of the 16 internal devices stored on the SmartServer.

The input parameters you supply to this function will include one or more <Item> elements with a LON_Device_Command_Invoke type and an attribute specifying the debugging command to be performed on the device. Each <Item> element will include the <UCPTname> of the device upon which a debugging command is to be issued.

QueryStatus

You can use the QueryStatus debugging command to test the performance of a device. The following example demonstrates how to query a device:

Request

```
<InvokeCmd xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Device_Command_Invoke" Command="QueryStatus">
        <UCPTname>Net/LON/iLON App</UCPTname>
    </Item>
  </iLonItem>
</InvokeCmd>
```

Response

```
<InvokeCmd xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
 <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="LON_Device_StatusData_InvokeResponse">
        <UCPTname>Net/LON/DIO-1</UCPTname>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-28T15:35:24.070-07:00</UCPTlastUpdate>
        <UCPTuniqueId>00a145791500</UCPTuniqueId>
        <UCPTtransmitErrors>0</UCPTtransmitErrors>
        <UCPTtransactionTimeouts>0</UCPTtransactionTimeouts>
        <UCPTrcvTransactionFull>0</UCPTrcvTransactionFull>
        <UCPTlostMessages>0</UCPTlostMessages>
        <UCPTmissedMessages>0</UCPTmissedMessages>
        <UCPTresetCause LonFormat="UCPTresetCause">DRC_SOFTWARE_RESET</UCPTresetCause>
        <UCPTversionNumber>100</UCPTversionNumber>
        <UCPTerrorLog LonFormat="UCPTerrorLog">DELT_NO_ERROR</UCPTerrorLog>
        <UCPTneuronModel LonFormat="UCPTneuronModel">MN_NEURON_3150</UCPTneuronModel>
        <UCPTonlineStatus LonFormat="UCPTonlineStatus">DST_CONFIGURED_ONLINE</UCPTonlineStatus>
    </Item>
  </iLonItem>
</InvokeCmd>
```

The QueryStatus command returns the following device statistics:

`<UCPTname>`	The name of the device in the following format: *<network>*/*<channel>*/*<device>*
`<UCPTuniqueId>`	The Neuron ID of the device as a 12-digit hex string. The Neuron ID is a unique 48-bit number burnt into the device's Neuron chip.
`<UCPTtransmitErrors>`	Transmission errors typically indicate cyclical redundancy check (CRC) errors. CRC errors are commonly caused by electromagnetic interference (EMI) on the channel.

<UCPTtransactionTimeouts>	Transaction timeouts occur when an acknowledged message times out after the last retry without the receiving device sending a confirmation that the message was delivered.
<UCPTrcvTransactionFull>	Transaction full errors occur when the device's transaction database, which is used to detect duplicate message packets, overflows. This may indicate excessive network traffic or transaction timers that are set too high.
<UCPTlostMessages>	Lost messages occur when a device's application buffer overflows. This may indicate excessive network traffic or a busy device application. If the incoming message is too large for the application buffer, an error is logged but the lost message count is not incremented.
<UCPTmissedMessages>	Missed messages occur when a device's network buffer overflows or network buffers are not large enough to accept all packets on the channel, whether or not addressed to this device.
<UCPTresetCause>	An error code that indicates the cause for the device's most recent reset. This property may be one of the following values: • DRC_CLEARED • DRC_POWER_UP • DRC_EXTERNAL_RESET • DRC_WATCHDOG_RESET • DRC_SOFTWARE_RESET
<UCPTversionNumber>	The firmware version used by the device hardware
<UCPTerrorLog>	Indicates whether errors have been logged for the device. Check the LonMaker Turbo Editions Help file to locate a description of the error.
<UCPTneuronModel>	Displays the model number of the device's Neuron chip (3120® or 3150®) or generic
<UCPTonlineStatus>	Indicates the status of the device and the device application. This property may be one of the following values: • DST_UNCONFIGURED • DST_APPLICATIONLESS • DST_CONFIGURED_ONLINE • DST_CONFIGURED_HARD_OFFLINE • DST_UNCONFIGURED_OFFLINE • DST_APPLICATIONLESS_OFFLINE • DST_CONFIGURED_SOFT_OFFLINE

Non-zero values indicate that the device was unable to receive and/or respond to a message. Small values are expected; rapidly increasing values may indicate a problem. If the device is consistently reporting failures and new errors are being logged, the device may have a configuration problem or the network may be overloaded.

ClearStatus

You can use the ClearStatus debugging command to clear the device statistics returned by the QueryStatus command. The following example demonstrates how to use this command:

```
<InvokeCmd xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Device_Command_Invoke" Command="ClearStatus">
        <UCPTname>Net/LON/iLON App</UCPTname>
    </Item>
  </iLonItem>
</InvokeCmd>
```

SendServicePin

You can use the SendServicePin command to send a service pin message from one of the 16 internal devices stored on the SmartServer. You can use this command to commission or re-commission an internal device on the SmartServer, such as the automated systems device (i.LON App) or an FPM application device, using an LNS application such as the LonMaker tool.

This command is useful because if you press the service pin on the SmartServer hardware when commissioning an internal device, it sends service pin messages from three of the internal devices defined on the SmartServer: i.LON App, the IP-852 router, and the SmartServer's network interface (i.LON NI). The following example demonstrates how to use this command on an internal SmartServer device:

```
<InvokeCmd xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Device_Command_Invoke" Command="SendServicePin">
        <UCPTname>Net/LON/HVAC FPM</UCPTname>
    </Item>
  </iLonItem>
</InvokeCmd>
```

14.3.4 Using the Delete Function on a LONWORKS Device

You can use the *Delete* function to delete a LONWORKS device on the SmartServer, or in an LNS network database via the LNS Proxy Web service. The *Delete* function takes an <Item> element with a LON_Device_Cfg type as its input. The <Item> element only needs to include the device's <UCPTname> property in the *Delete* Request as demonstrated in the following code sample:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Device_Cfg">
        <UCPTname>MyOldNetwork/MyOldChannel/MyOldDevice</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname> MyOldNetwork/MyOldChannel/MyOldDevice</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

14.4 Routers

You can use the *List*, *Get*, *Set*, and *Delete* functions on routers as described in the previous section, *LONWORKS Devices*. Routers have the same properties as devices except for the following differences:

- A router has the xsi type LON_Device_Router_Cfg.

- The <UCPTannotation> property is Router.
- The <UCPTuri> property is LON_Device_Router_Cfg.htm.

Note that LON_Device_Router_Cfg has three additional properties:

`<Device>`	The network path of the channel attached to the far side of the router in the following format: <*network*>/<*channel*>/<*router*>.
`<UCPTrouterClass>`	• This property specifies one of the following seven router types: • LCA_CONFIGURED_ROUTER. The router determines which packets to forward based on internal routing tables. These routing tables contain one entry for each subnet in the application domain. Whenever a router receives a packet, it examines the source and destination subnet ID to determine whether to forward the packet. This is the recommended type because it optimizes network traffic and enables the channels on which devices are attached to be determined automatically. Configured routers also support the use of redundant routers (multiple routers connecting two channels), which provide for redundant message paths and greater system reliability. • LCA_LEARNING_ROUTER. Like a configured router, the router determines which packets to forward based on internal routing tables. Learning routers, though, have their routing tables stored in volatile memory; therefore, the router forwards packets addressed to all subnets in the application domain after being reset. Whenever a learning router receives a packet from one of its channels, it uses the source subnet ID to learn the network topology. It sets the corresponding routing table entries to indicate that the subnet in question is to be found in the direction from which the packet was received. A learning router always forwards all group-addressed messages. • LCA_REPEATER. The router forwards all valid packets received on one channel to the other channel. Subnets cannot span non-permanent repeaters. You can use a non-permanent repeater to maintain flexibility in order to change the router type later. This is the default. • LCA_BRIDGE. The router forwards all valid packets that match the network domain. Subnets cannot span non-permanent bridges. You can use a non-permanent bridge to maintain flexibility in order to change the router type later. • LCA_PERMANENT_REPEATER. The router behaves like a repeater, except that you cannot change the router type after the router has been created. Subnets may span permanent repeaters. You can use permanent repeaters to preserve subnet IDs. • LCA_PERMANENT_BRIDGE. The router behaves like a bridge, except that you cannot change the router type after the router has been created. Subnets may span permanent

	bridges. You can use permanent bridges to preserve subnet IDs. • LCA_NUL. The SmartServer automatically select the appropriate router type.
<UCPTport>	The port on the SmartServer used for receiving messages from the IP-852 Configuration Server. The default port is **1628**.

14.5 Remote Network Interface

You can use the *List*, *Get*, *Set*, and *Delete* functions on a remote network interface (RNI) as described in the previous section, *LONWORKS Devices*. An RNI has the same properties as devices except for the following differences:

- An RNI has an xsi type of LON_Device_RNI_Cfg.
- The <UCPTannotation> property is RNI.
- The <UCPTuri> property is LON_Device_RNI_Cfg.htm.

Note that LON_Device_RNI_Cfg has three additional properties:

<UCPTport>	The port on the SmartServer used to listen for LonTalk packets when it is being used as an RNI. The default port is **1628**.
<UCPTmaxRxTransactions>	The maximum number of receive transactions that the RNI application can receive at one time. This value may range from 1 to 32,768. The default value is **16**. You can increase this value if you observe buffer overflows. You can decrease this value if you observe that the SmartServer's memory is low. A receive transaction entry is required for any incoming message which uses either repeating or acknowledged messaging service (a receive transaction is not required for messages using unacknowledged service). A receive transaction entry is also required for each unique source address/destination address/priority attribute. Each receive transaction entry contains a current transaction number. A message is considered to be a duplicate if its source address, destination address, and priority attribute vector into an existing receive transaction and the message's transaction number matches the entry's transaction number. Receive transaction entries are freed after the receive timer expires. The receive timer duration is determined by the destination device and varies as a function of the message addressing mode. For group addressed messages, the receive timer is determined by the address table. For Neuron ID addressed messages, the receive timer is fixed at eight seconds. For other addressing modes, the non-group receive timer in the configuration data structure is used.

14.6 LONWORKS Functional Blocks

The following section describes how to use the *List*, *Get*, *Set*, and *Delete* functions on LONWORKS functional blocks.

14.6.1 Using the List Function on a LONWORKS Functional Block

You can use the *List* function to retrieve a list of functional blocks on the SmartServer or to retrieve a list of functional blocks in a specific LNS network database via the LNS Proxy Web service. The List function takes an <iLonItem> element that has an xSelect statement with a LON_Fb_Cfg type as its input, as shown in the example below.

Request (return all the functional blocks of all the devices on the SmartServer or in an LNS network database)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="LON_Fb_Cfg"]</xSelect>
  </iLonItem>
</List>
```

Alternatively, you can filter the functional blocks returned by the *List* function to those on a specific device by including the <UCPTname>of the parent device in the xSelect statement, or you can filter functional blocks returned by using the <UCPTname>, <UCPTlastUpdate>, or the <UCPThidden> properties of the functional block.

Request (return all the functional blocks of a specific device that were updated after a specific time)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="LON_Fb_Cfg"][starts-with(UCPTname,"Building 2/Channel 1/DIO-5")]
                  [UCPTlastUpdate&gt;"2008-03-31T00:00:00"]
    </xSelect>
  </iLonItem>
</List>
```

Request (return all the functional blocks of a specific type on all devices based on name)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="LON_Fb_Cfg"][contains(UCPTname,"Digital Output")]</xSelect>
  </iLonItem>
</List>
```

Request (use an xSelect statement return all functional blocks on a specific device that are currently hidden in the SmartServer Web interface)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect> //Item[@xsi:type="LON_Fb_Cfg"][starts-with(UCPTname,"Building 2/Channel 1/iLON
              App")][UCPThidden="1"]
    </xSelect>
  </iLonItem>
</Get>
```

Request (return all the functional blocks of a specific type based on name)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="LON_Fb_Cfg"][starts-with(UCPTname,"Building 2/Channel 1/DIO-5/Digital
    Output")]</xSelect>
  </iLonItem>
</List>
```

Response

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
    <Item>
        <UCPTname>Building 2/Channel 1/DIO-5/Digital Output[0]</UCPTname>
        <UCPTannotation>#8000010000000000[3].UFPTDigitalOutput;xsi:type="LON_Fb_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
```

```
    <Item>
        <UCPTname>Building 2/Channel 1/DIO-5/Digital Output[1]</UCPTname>
        <UCPTannotation>#8000010000000000[3].UFPTDigitalOutput;xsi:type="LON_Fb_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
  </iLonItem>
</List>
```

14.6.2 Using the Get Function on a LONWORKS Functional Block

You can use the *Get* function to retrieve the configuration of a functional block defined on the SmartServer or in a specific LNS network database. The input parameters you supply to this function will include one or more <Item> elements with a LON_Fb_Cfg type. Each <Item> element will include the <UCPTname> of each functional block whose configuration is to be returned by this function, as shown in the example below.

Request (return a specific functional block)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Fb_Cfg">
        <UCPTname>Building 2/Channel 1/DIO-5/Digital Output[0]</UCPTname>
    </Item>
  </iLonItem>
</Get>
```

Alternatively, you can specify one or more functional block properties in the xSelect statement to filter the items returned by the *Get* function, including the <UCPTname> to filter functional blocks based on their parent device, and the <UCPTfptKey> to filter functional blocks based on the functional profile template.

Request (use an xSelect statement return all the functional blocks on a specific LONWORKS device)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect> //Item[@xsi:type="LON_Fb_Cfg"][starts-with(UCPTname,"Building 2/Channel
    1/DIO-5/")]</xSelect>
  </iLonItem>
</Get>
```

Request (use an xSelect statement to return all functional blocks from a specific manufacturer based on the functional profile template)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect> //Item[@xsi:type="LON_Fb_Cfg"][UCPTfptKey="#8000010000000000[3].UFPTDigitalCounter"]
    </xSelect>
  </iLonItem>
</Get>
```

Response

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="LON_Fb_Cfg">
        <UCPTname>Building 2/Channel 1/DIO-5/Digital Counter[0]</UCPTname>
        <UCPTannotation>#8000010000000000[3].UFPTDigitalCounter;xsi:type="LON_Fb_Cfg"
        </UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-31T12:33:52.013-07:00</UCPTlastUpdate>
        <UCPTuri>LON_Fb_Cfg.htm</UCPTuri>
        <UCPTfbIndex>1</UCPTfbIndex>
        <UCPTfptKey>#8000010000000000[3].UFPTDigitalCounter</UCPTfptKey>
        <UCPTdynamic LonFormat="UCPTdynamic">DDT_STATIC</UCPTdynamic>
    </Item>
    <Item xsi:type="LON_Fb_Cfg">
        <UCPTname>Building 2/Channel 1/DIO-5/Digital Counter[1]</UCPTname>
        <UCPTannotation>#8000010000000000[3].UFPTDigitalCounter;xsi:type="LON_Fb_Cfg"
        </UCPTannotation>
```

```
      <UCPThidden>0</UCPThidden>
      <UCPTlastUpdate>2008-03-31T12:33:54.133-07:00</UCPTlastUpdate>
      <UCPTuri>LON_Fb_Cfg.htm</UCPTuri>
      <UCPTfbIndex>2</UCPTfbIndex>
      <UCPTfptKey>#80000100000000000[3].UFPTDigitalCounter</UCPTfptKey>
      <UCPTdynamic LonFormat="UCPTdynamic">DDT_STATIC</UCPTdynamic>
  </Item>
 </iLonItem>
```

The *Get* function returns an <Item> element for each functional block referenced in the input parameters you supplied to the function. The properties included within each <Item> element are initially defined when the functional block is added to the SmartServer or LNS network database. You can write to these functional block properties with the *Set* function. The following table describes these properties.

Property	Description
`<UCPTname>`	The name of the functional block in the following format: <*network*/*channel*/*device*/*functionalblock*>. You can rename a functional block by providing its <UCPTfbIndex> and specifying the new <UCPTname> property to which the functional block is to be renamed.
`<UCPTannotation>`	The type of functional block (**DefaultFb** by default) and its xsi type, which is LON_Fb_Cfg. This determines the icon used to represent the functional block in the SmartServer Web interface. You can use custom icons (.gif images) to represent your company's functional blocks in the SmartServer Web interface. If a custom icon is being used for a functional block, this property is set to <*manufacturer ID*>**[***scope selector***].** <*functional profile programmatic name*>. To use a custom functional block icon, upload the <*manufacturer ID*>**[***scope selector***].** <*functional profile programmatic name*>.**gif** file for your custom functional block icon to both the root/Web/images/tree and the root/Web/images/app folders on the SmartServer flash disk. The root/web/images/tree stores the icons shown in the SmartServer tree. The root/web/images/app folder stores the icons shown in the upper left-hand corner of an object's configuration and driver Web pages.
`<UCPThidden>`	A flag indicating whether the functional block is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **`0`** – shown **`1`** – hidden
`<UCPTlastUpdate>`	A timestamp indicating the last time the configuration of the functional block was updated. This timestamp uses the ISO 8601 format, which is as follows: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Data Point was last updated. The second segment (after the **T):** HH:MM:SS.sss represents the time of day the configuration of the Data Point

Property	Description
	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 be expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. It can be left out. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002. If it is not UTC, time shift has to be defined: 2008-02-28T09:59:53.660+01:00
`<UCPTdescription>`	A user-defined description of the functional block. This can be a maximum of 201 characters long.
`<UCPTuri>`	The name of the file on the SmartServer flash disk containing the configuration of the functional block. This property is LON_Fb_Cfg.htm by default.
`<DataPoint>`	The input and output data points encapsulated by the functional block are signified by an array of <DataPoint> elements. Each <DataPoint> element includes "dpType" and "discrim" attributes signifying the data point's programmatic name, as defined by the functional profile template, and the data point's direction. Each <DataPoint> element contains the following two properties: • <UCPTName>. The name of the data point in the following format: <*network*/*channel*/*device*/*functional block*/*data point*>. • <UCPTformatDescription>. The data point's program ID; data type (SNVT, SCPT, UNVT, UCPT, or built-in data type); and format (e.g., SI metric or US customary if the type has multiple formats such as **SNVT_temp_p**). The format description is displayed in the following format: • #<manufacturer ID>**[**scope selector**].**<type name>[#format]
`<UCPTfbIndex>`	The index number of the functional block within its associated device. When you use the *Set* function to modify the configuration of an existing functional block, you must specify the functional block's index.
`<UCPTfptKey>`	The functional profile that is valid for this functional block in the following format: **#**<*device program ID*>**[***scope selector***].** <f*unctional profile name*>. This field is read-only. The scope selector specifies the context in which the network variables and configuration properties within a functional block are interpreted. The scope selector may be any of the following

Property	Description
	values: • **0**. Standard functional profile defined in the standard resource file set. • **3**. User-defined functional profile, defined in a manufacturer-specific resource file set. • **4**. User-defined functional profile, defined in a manufacturer and device class specific resource file set. • **5**. User-defined functional profile, defined in a manufacturer and device class/subclass specific resource file set. **6**. User-defined functional profile, defined in manufacturer, and device class/subclass/model number specific resource file set. This property cannot be null for dynamic functional blocks.
<UCPTdynamic>	Indicates whether the functional block is static (DDT_STATIC) or dynamic (DDT_DYNAMIC). You cannot use the *Set* function to modify this property.

14.6.3 Using the Set Function on a LONWORKS Functional Block

You can use the *Set* function to overwrite the configuration of a functional block, or to create a new functional block. The input parameters you supply to the function will include one or more <Item> elements. Each <Item> element includes a <UCPTname> property that specifies a unique functional block to be created or modified.

Each <Item> element may also include a series of properties that define the configuration of the new (or modified) functional block. This set of properties is the same whether you are creating a new functional block or modifying an existing functional block.

- If you are creating a new functional block, you need to specify the <UCPTfbIndex> and <UCPTfptKey> properties; all other properties are optional.
- If you are modifying an existing functional block, you must specify the <UCPTfbIndex> property. In addition, all other properties should be filled; otherwise the values stored in them are erased. The previous section, *Using the Get Function on a LonWorks Functional Block*, details the properties you can include in the *Set* function.

You can set multiple functional blocks with a single *Set* message. However, you should not attempt to create or write to more than 100 functional blocks with a single call to the *Set* function. The following example demonstrates how to create a new functional block.

Request (add a functional block to a device)

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>Building 2/Channel 1/DIO-5/Digital Counter[0]</UCPTname>
        <UCPTfbIndex>1</UCPTfbIndex>
        <UCPTfptKey>#80000100000000000[3].UFPTDigitalCounter</UCPTfptKey>
    </Item>
  </iLonItem>
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
```

```
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname> Building 2/Channel 1/DIO-5/Digital Counter[0]</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

14.6.4 Using the Delete Function on a LONWORKS Functional Block

You can use the *Delete* function to delete a functional block on the SmartServer, or in an LNS network database via the LNS Proxy Web service. The *Delete* function takes an <Item> element with a LON_Fb_Cfg type as its input. The <Item> element only needs to include the functional block's <UCPTname> property in the *Delete* Request as demonstrated in the following code sample:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Fb_Cfg">
        <UCPTname>MyOldNetwork/MyOldChannel/MyOldDevice/MyOldFb</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>MyOldNetwork/MyOldChannel/MyOldDevice/MyOldFb</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

14.7 Network Variables (LONWORKS Data Points)

The following section describes how to use the *List*, *Get*, *Set*, and *Delete* functions on network variables (LONWORKS data points). For information on reading and writing values to network variables, see Chapter 4, *Using the SmartServer Data Server*.

14.7.1 Using the List Function on Network Variables

You can use the *List* function to retrieve a list of network variables (LONWORKS data points) on the SmartServer or to retrieve a list of network variables in a specific LNS network database via the LNS Proxy Web service. The List function takes an <iLonItem> element that has an xSelect statement with a LON_Dp_Cfg type as its input, as shown in the example below.

Request (return all the Network variables of all the functional blocks on the SmartServer or in an LNS network database)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="LON_Dp_Cfg"]</xSelect>
  </iLonItem>
</List>
```

Alternatively, you can filter the network variables (LONWORKS data points) returned by the *List* function to those on a specific functional block by including the <UCPTname>of the parent functional block in the xSelect statement, or you can filter the network variables returned using the <UCPTname> and <UCPTlastUpdate> data point properties.

Request (use an xSelect statement to return all the network variables on a specific functional block)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
```

```
    <xSelect>//Item[@xsi:type="LON_Dp_Cfg"][starts-with(UCPTname,"Building 2/Channel 1/DIO-5/Digital
    Output[0]")]
    </xSelect>
  </iLonItem>
</List>
```

Request (use an xSelect statement to return all the network variables that were updated after a specific time)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="LON_Dp_Cfg"][UCPTlastUpdate&gt;"2008-03-31T00:00:00"]
    </xSelect>
  </iLonItem>
</List>
```

Request (return all the network variables of a specific type based on name)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="LON_Dp_Cfg"][contains(UCPTname,"DO_Digital")]</xSelect>
  </iLonItem>
</List>
```

Response

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
    <Item>
        <UCPTname>Building 2/Channel 1/DIO-5/Digital Output[0]/DO_Digital_1</UCPTname>
        <UCPTannotation>Dp_In;xsi:type="LON_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
    <Item>
        <UCPTname>Building 2/Channel 1/DIO-5/Digital Output[1]/DO_Digital_2</UCPTname>
        <UCPTannotation>Dp_In;xsi:type="LON_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
  </iLonItem>
</List>
```

14.7.2 Using the Get Function on Network Variables

You can use the *Get* function to retrieve the configuration of network variables (LONWORKS data points) defined on the SmartServer or in a specific LNS network database. The input parameters you supply to this function will include one or more <Item> elements with a LON_Dp_Cfg type. Each <Item> element will include the <UCPTname> of each network variable whose configuration is to be returned by this function, as shown in the example below.

Request (use an xSelect statement to return a specific network variable)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Dp_Cfg">
        <UCPTname>Building 2/Channel 1/DIO-5/Digital Output[0]/DO_Digital_1/</UCPTname>
    </Item>
  </iLonItem>
</Get>
```

Alternatively, you can specify one or more network variable (LONWORKS data point) properties in the xSelect statement to filter the items returned by the *Get* function, including the <UCPTname> to filter data points based on their parent functional block.

Request (use an xSelect statement return all the network variables on a specific functional block)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect> //Item[@xsi:type="LON_Fb_Cfg"][starts-with(UCPTname,"Building 2/Channel 1/DIO-5/
    Digital Output[0]")]</xSelect>
```

```
  </iLonItem>
</Get>
```

Response

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="LON_Dp_Cfg">
        <UCPTname>Building 2/Channel 1/DIO-5/Digital Output[0]/DO_Digital_1</UCPTname>
        <UCPTannotation>Dp_In;xsi:type="LON_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-31T12:34:28.483-07:00</UCPTlastUpdate>
        <UCPTdescription>Digital value to output</UCPTdescription>
        <UCPTuri>LON_Dp_Cfg.htm</UCPTuri>
        <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
        <UCPTlength>2</UCPTlength>
        <UCPTdirection LonFormat="UCPTdirection">DIR_IN</UCPTdirection>
        <UCPTpersist>0</UCPTpersist>
        <UCPTunit field="value">% of full level</UCPTunit>
        <UCPTunit field="state">state code</UCPTunit>
        <UCPTnvIndex>12</UCPTnvIndex>
        <UCPTnvSelector>3ff3</UCPTnvSelector>
        <UCPTdynamic LonFormat="UCPTdynamic">DDT_STATIC</UCPTdynamic>
        <UCPTpollRate>600.0</UCPTpollRate>
    </Item>
  </iLonItem>
```

The *Get* function returns an <Item> element for each network variable (LONWORKS data point) referenced in the input parameters you supplied to the function. The properties included within each <Item> element are initially defined when the network variable is added to the SmartServer or LNS network database. You can write to these data point properties with the *Set* function. The following table describes these properties.

Property	Description
<UCPTname>	The name of the network variable in the following format: <*network*/*channel*/*device*/*functionalblock*/*data point*>. You can rename a network variable by providing its <UCPTnvIndex> and specifying the new <UCPTname> property to which the network variable is to be renamed.
<UCPTannotation>	The type of network variable and its xsi type, which is LON_Dp_Cfg. This determines the icon used to represent the network variable in the SmartServer Web interface.
<UCPThidden>	A flag indicating whether the network variable is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **0** – shown **1** – hidden
<UCPTlastUpdate>	A timestamp indicating the last time the configuration of the network variable was updated. This timestamp uses the ISO 8601 format, which is as follows: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Data Point was last updated. The second segment (after the **T):** HH:MM:SS.sss represents the time of day the configuration of the Data Point

Property	Description
	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 be expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. It can be left out. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002. If it is not UTC, time shift has to be defined: 2008-02-28T09:59:53.660+01:00
`<UCPTdescription>`	A user-defined description of the network variable. This can be a maximum of 201 characters long.
`<UCPTuri>`	The name of the file on the SmartServer flash disk containing the configuration of the network variable. This property is LON_Dp_Cfg.htm by default.
`<UCPTformatDescription>`	• The network variable's program ID; data type (SNVT, SCPT, UNVT, UCPT, or built-in data type); and format (e.g., SI metric or US customary if the type has multiple formats such as **SNVT_temp_p**). The format description is displayed in the following format: #<*manufacturer ID*>**[***scope selector***].**<*type name*>[#format] . This determines many factors about the network variable, including the type of values it takes and its base type. This could be any standard (SNVT) format type included in the resource files on the SmartServer, or any user-defined (UNVT) format type included in resource files uploaded to the SmartServer. For more information on the resource files, see *SmartServer Resource Files*. If you do not set this property, it is set to RAW_HEX and the network variable uses raw hex values. The SNVT format types included in the SmartServer resource files are also listed and described in the SNVT Master List, which can be downloaded from Echelon's Support Web site at: *www.echelon.com*
`<UCPTlength>`	Specifies the size (in bytes) of the network variable.
`<UCPTdirection>`	Specifies whether the network variable is an input data point (DIR_IN), output data point (DIR_OUT), or has an unspecified direction (DIR_NUL).
`<UCPTpersist>`	A flag indicating that the value stored in the network variable persists through SmartServer reboots. If this property is set to 1, the last network variable value is stored in the

Property	Description
	<UCPTdefOutput> property.
<UCPTunit>	This property is a string up to 227 characters long that describes the units the value in which a network variable is measured. It is based on the data type assigned to the network variable. 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 resource files on the SmartServer. For scalar and enumerated data points, this property specifies the units of measures used by the data point. For example, the unit string of a **SNVT_temp_f** data point is **°F**. The unit string is defined by resource files. For structured data points, the fields within the data point are specified in a series of <UCPTunit> properties if the unit strings can be edited. Using a **SNVT_switch** data point for example, **value** and **state** fields will be specified in a series of <UCPTunit> properties with their respective units of measure ("% of full level" and "state code"). You can use the *Set* function to edit the unit strings of data point fields.
<UCPTnvIndex>	The index number of the network variable within its device. You cannot use the *Set* function on this property.
<UCPTnvSelector>	Displays the value that uniquely associates the network variable with its connections. If the network variable is not a member of a connection, the selector is set to a value representing an unbound network variable. For LONWORKS connections, a selector is a 14-bit number used to identify connected network variable. When placing the network variable in a LONWORKS connection, the SmartServer assigns the network variable a value representing that connection. All network variables in a given connection use the same selector. The LNS Server shares a network variable selector among connections if the connections share one or more data points. You cannot use the *Set* function on this property.
<UCPTdynamic>	Indicates whether the network variable is static (DDT_STATIC), dynamic (DDT_DYNAMIC), or is changeable (DDT_CHANGEABLE). You cannot use the *Set* function to modify this property.

Property	Description
<UCPTpollRate>	The frequency in which the SmartServer's internal data server polls the network variable. The recommended minimum poll rate is 30 seconds; the maximum poll rate is 1 second. The default poll rate for network variables is **600** seconds. You should set poll rates for the data points of the external devices that are connected to the SmartServer. **Note**: The actual poll rate is determined by calculating the least common denominator of all the poll rates set for the data point from the applications to which it has been added.

14.7.3 Using the Set Function on a Network Variable

You can use the *Set* function to overwrite the configuration of a network variable, or to create a new network variable. The input parameters you supply to the function will include one or more <Item> elements. Each <Item> element includes a <UCPTname> property that specifies a unique network variable to be created or modified.

Each <Item> element may also include a series of properties that define the configuration of the new (or modified) network variable. This set of properties is the same whether you are creating a new network variable or modifying an existing network variable.

- If you are creating a new network variable, you need to specify the <UCPTnvIndex> and <UCPTformatDescription> properties; all other properties are optional.
- If you are modifying an existing network variable, you must specify the <UCPTnvIndex> property. In addition, all other properties should be filled; otherwise the values stored in them are erased. The previous section, *Using the Get Function on Network Variables*, details the properties you can include in the *Set* function.

You can set multiple network variables with a single *Set* message. However, you should not attempt to create or write to more than 100 network variables with a single call to the *Set* function. The following example demonstrates how to create a new network variable.

Request (add a data point to a functional block)

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item>
        <UCPTname>Building 2/Channel 1/DIO-5/Digital Output[0]/DO_Digital_1</UCPTname>
        <UCPTnvIndex>1</UCPTfbIndex>
        <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
    </Item>
  </iLonItem>
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Building 2/Channel 1/DIO-5/Digital Output[0]/DO_Digital_1</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

14.7.4 Using the Delete Function on a Network Variable

You can use the *Delete* function to delete a network variable on the SmartServer, or in an LNS network database via the LNS Proxy Web service. The *Delete* function takes an <Item> element with

a LON_Dp_Cfg type as its input. The <Item> element only needs to include the network variable's <UCPTname> property in the *Delete* Request as demonstrated in the following code sample:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="LON_Dp_Cfg">
        <UCPTname>MyOldNetwork/MyOldChannel/MyOldDevice/MyOldFb/MyOldNv</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>MyOldNetwork/MyOldChannel/MyOldDevice/MyOldFb/MyOldNv</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

14.8 Configuration Properties (LONWORKS Data Points)

You can use the *List*, *Get*, *Set*, and *Delete* functions on configuration properties as described in the previous section, *Network Variables (LONWORKS Data Points)*. For information on reading and writing values to configuration properties, see Chapter 4, *Using the SmartServer Data Server*.

Configuration properties have the same properties as network variables except for the following differences:

- A configuration property has an xsi type of LON_Cp_Dp_Cfg. A configuration property file has an xsi type of LON_Cp_File_Cfg.
- The <UCPTuri> property is LON_Cp_Dp_Cfg.htm for configuration properties and LON_Cp_File_Cfg for configuration property files.
- The default poll rate for configuration properties is **0** seconds, which means that means polling is disabled. You must set a poll rate for configuration properties to update their values via polling.

Note that LON_Cp_File_Cfg includes an additional <UCPThandle> property, and both LON_Cp_Dp_Cfg and LON_Cp_File_Cfg have a set of read-only restriction flags that determine whether a configuration property value can be changed in a given scenario. The values of these restriction flags are determined by the device interface (XIF) file. They cannot be modified with the *Set* function. The following table describes these additional CP properties:

<UCPThandle>	This property is applicable to configuration property files only (LON_Cp_File_Cfg). The handle of the configuration property file assigned by the LNS server. You cannot use the *Set* function to modify the handle assigned to the configuration property file. Note that you cannot rename configuration property files.
<UCPTreadOnlyFlag>	The configuration property is a constant; its value cannot be changed.
<UCPTdeviceFlag>	The value of the configuration property is always read from the device and can be modified independent of the LNS database.
<UCPTmanufactureFlag>	The configuration property value can only be changed when the device is being licensed.

<UCPTobjDisableFlag>	The device must be disabled for the configuration property value to be changed.
<UCPTofflineFlag>	The device must be offline for the configuration property value to be changed.
<UCPTresetFlag>	The device is reset after the configuration property value is changed.
<UCPToffset>	This property is applicable to configuration property files only (LON_Cp_File_Cfg). The offset in the file with the given index where the configuration property begins.
<UCPTfileIndex>	This property is applicable to configuration property files only (LON_Cp_File_Cfg). The file index on the device in which the configuration property exists.
<UCPTrelation>	If the configuration property applies to a specific network variable, displays network path of the network variable.

14.9 LONWORKS Connections

You can use the LNS Proxy Web service to create LONWORKS connections that bind the network variables of LONWORKS devices that are in the same LNS network database on the same LNS Server. Creating LONWORKS connections is similar to creating Web connections as they use the same SOAP interface. You use the same *List*, *Get*, *Set*, and *Delete* functions as described in the *Using the Web Binder Application* section in Chapter 4.

LONWORKS connections have the same properties as Web connections except for the following differences with the <DataPoint> property referencing the target data point in the LONWORKS connection:

- You can specify one of the following options for the <UCPTserviceType>. These service types vary in reliability and resources consumed:
  - ST_LON_ACK (Acknowledged). The sending device expects to receive confirmation from the receiving device or devices that a network variable update was delivered. The sending application is notified when an update fails, but it is up to the developer of the sending device to handle the notification in the device application.

    While acknowledged service is very reliable, it can create excessive message traffic, especially for large fan-out or polled fan-in connections. When acknowledged messaging is used, every receiving device has to return an acknowledgment.

    Acknowledged messaging can be used with up to 63 receiving devices, but an acknowledged message to 63 devices generates at least 63 acknowledgements—more if any retries are required due to lost acknowledgements.
  - ST_LON_REPEATED (Repeated). The sending device sends out a series of network variable updates, but does not expect any confirmation from the receiving device. Repeated service with three repeats has a 99.999% success rate in delivering messages.

    Repeated service provides the same probability of message delivery as acknowledged messaging with the same number of retries, with significantly lower network overhead for large multicast fan-out connections.

    For example, a repeated message with three retries to 64 devices generates four packets on the network, whereas an acknowledged message requires at least 64 packets.

- ST_LON_UNACK (Unacknowledged). The sending device sends out the network variable update only once and does not expect any confirmation from the receiving device. This message service type consumes the least amount of resources, but is the least reliable.
- ST_NUL (Unknown). The SmartServer selects the service type.

- The <UCPTservicePath> property is always set to //WebService[UCPTindex=0], where 0 is referring to the LNS Server containing the LNS network database in which the source and target network variables are stored.
- The <UCPTpropagate> property is set to 1 by default and cannot be modified with the *Set* function.

The following example demonstrates how to use the *Set* function create a LONWORKS connection between an internal SmartServer device and an external device connected to the SmartServer. When a *Write* function is performed on the source data point in the Web connection (Building 2/Channel 1/iLON App/Digital Input 2/nvoClsValue_2), the updated value is propagated to the target data points (Building 2/Channel 1/DIO-1/Digital Output[0]/DO_Digital_1) in the LONWORKS connection.

Request

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="Dp_Ref">
        <UCPTname>Building 2/Channel 1/iLON App/Digital Input 2/nvoClsValue_2</UCPTname>
        <DataPoint dpType="Target">
            <UCPTname>Building 2/Channel 1/DIO-1/Digital Output[0]/DO_Digital_1</UCPTname>
            <UCPTserviceType xsi:type="string" LonFormat="UCPTserviceType">ST_LON_ACK
            </UCPTserviceType>
            <UCPTservicePath xsi:type="string">//WebService[UCPTindex=0]</UCPTservicePath>
            <UCPTpriority>240</UCPTpriority>
        </DataPoint>
    </Item>
  </iLonItem>
</Set>
```

Note: All LONWORKS connections created with the LNS Proxy Web service use subnet/node ID addressing. This means that a message packet travels from the sending device to the destination device using the 2-byte logical address of the destination device in the network. Overall, LONWORKS connections use the following connection options:

Service Type	Acknowledged (the default), Repeated, or Unacknowledged. See the previous section for specifying the service type in the <UCPTserviceType> property.
Addressing	Subnet/Node ID.
Priority	Used if hub (source) network variable specifies priority.
Authentication	Used if target network variable has authentication enabled.
Retry Count	Calculated based on topology and service type.
Repeat Count	Calculated based on topology and service type.
Repeat Timer	Calculated based on topology and service type.
Receive Timer	Calculated based on topology and service type.

Transaction Timer	Calculated based on topology and service type.
Broadcast Options	Broadcast addressing is not used.
Alias Options	Network variable aliases are used to resolve selector conflicts.

15 Modbus Driver

The following chapter describes how to manage Modbus channels, devices, and data points on the SmartServer.

15.1 Modbus Channels

The following section describes how to use the *List*, *Get*, *Set*, and *Delete* functions on Modbus channels.

15.1.1 Using the List Function on Modbus Channels

You can use the *List* function to retrieve a list of Modbus channels on the SmartServer. The List function takes an <iLonItem> element that has an xSelect statement with a MOD_Channel_Cfg type as its input, as shown in the example below.

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="MOD_Channel_Cfg"]</xSelect>
  </iLonItem>
</List>
```

Response

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
    <Item>
        <UCPTname>Net/Modbus</UCPTname>
        <UCPTannotation>RS485;xsi:type="MOD_Channel_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
  </iLonItem>
</List>
```

The *List* function returns a list of <Item> elements for each Modbus channel defined on the SmartServer. You could use the list of <Item> elements returned by this function as input for the *Get* function. The *Get* function would then return the configuration of each Modbus channel included in the list. The next section describes the properties included in each of these elements.

15.1.2 Using the Get Function on Modbus Channels

You can use the *Get* function to retrieve the configuration of a Modbus channel defined on the SmartServer. The input parameters you supply to this function will include one or more <Item> elements with a MOD_Channel_Cfg type. Each <Item> element will include the <UCPTname> of each channel whose configuration is to be returned by this function, as shown in the example below.

Request

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="MOD_Channel_Cfg">
        <UCPTname>Net/Modbus Channel</UCPTname>
    </Item>
  </iLonItem>
</Get>
```

Response

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="MOD_Channel_Cfg">
        <UCPTname>Net/Modbus Channel</UCPTname>
```

```
        <UCPTannotation>RS232</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-04-02T11:07:27.860-07:00</UCPTlastUpdate>
        <UCPTuri>MOD_Channel_Cfg.htm</UCPTuri>
        <UCPThandle>0</UCPThandle>
        <UCPTchannelType LonFormat="UCPTchannelType">CT_RS232_MASTER</UCPTchannelType>
        <InterfaceOptions>
            <UCPTspeed LonFormat="UCPTspeed">MS_9600</UCPTspeed>
            <UCPTsize LonFormat="UCPTsize">CS_8</UCPTsize>
            <UCPTparity LonFormat="UCPTparity">P_NONE</UCPTparity>
            <UCPTstopBits LonFormat="UCPTstopBits">SB_1</UCPTstopBits>
        </InterfaceOptions>
        <UCPTserialMode LonFormat="UCPTserialMode">SM_RTU</UCPTserialMode>
        <UCPTretryCount LonFormat="UCPTretryCount">1</UCPTretryCount>
    </Item>
  </iLonItem>
</Get>
```

The *Get* function returns an <Item> element for each channel referenced in the input parameters you supplied to the function. The properties included within each <Item> element are initially defined when the channel is added to the SmartServer. You can write to these channel properties with the *Set* function. The following table describes these properties.

Property	Description
`<UCPTname>`	The name of the channel in the following format: <*network*/*channel*>. You can rename a Modbus channel by providing its <UCPThandle> and specifying the new <UCPTname> property to which the channel is to be renamed.
`<UCPTannotation>`	The type of Modbus channel, which may be IP, RS232, or RS485. This determines the icon used to represent the Modbus channel in the SmartServer Web interface.
`<UCPThidden>`	A flag indicating whether the Modbus channel is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **`0`** – shown **`1`** – hidden
`<UCPTlastUpdate>`	A timestamp indicating the last time the configuration of the Modbus channel was updated. This timestamp uses the ISO 8601 format, which is as follows: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Data Point was last updated. The second segment (after the **T):** HH:MM:SS.sss represents the time of day the configuration of the Data Point 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 be expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. It can be left out. For example, 2002-08-15T10:13:13Z indicates a UTC time of

Property	Description
	10:13:13 AM on August 15, 2002. If it is not UTC, time shift has to be defined: 2008-02-28T09:59:53.660+01:00
<UCPTdescription>	A user-defined description of the channel. This can be a maximum of 201 characters long.
<UCPTuri>	The name of the file on the SmartServer flash disk containing the configuration of the Modbus channel. This property is always MOD_Channel_Cfg.htm.
<UCPThandle>	The handle assigned to the Modbus channel assigned by the SmartServer. When you use the *Set* function to modify the configuration of an existing Modbus channel, you must specify the channel's handle. If you do not specify the handle, a new channel is created. You cannot use the *Set* function to modify the handle assigned to the channel.
<UCPTchannelType>	The channel type. This property may be one of the following values: • CT_RS485_MASTER. RS-485 is a balanced line, half-duplex system that allows transmission distances of up to 1.2 km. RS-485 allows for transmission over longer distances at higher speeds. This is the default. • CT_RS232_MASTER. RS-232 uses serial binary data for transmitting data between two devices. • CT_TCP_IP_MASTER. Modbus messages are enveloped in TCP/IP packets. TCP/IP allows for more versatile network systems, as Modbus connection can co-exists with other types of connections.
<UCPTport>	The port used on the SmartServer for listening for messages from the Modbus driver. The default port is 502. This property is available for TCP/IP Modbus channels only (<UCPTchannelType> is CT_TCP_IP_MASTER).

Property	Description
<InterfaceOptions>	This element contains the following options for Modbus channels attached to the RS-232 or RS-485 serial ports on the SmartServer (<UCPTchannelType> is CT_RS485_MASTER or CT_RS232 _MASTER): • <UCPTspeed>. The baud rate at which the SmartServer communicates with the Modbus devices on the channel. The default value is MS_9600. This property may be one of the following values: ▪ MS_110 ▪ MS_300 ▪ MS_600 ▪ MS_1200 ▪ MS_2400 ▪ MS_4800 ▪ MS_9600 ▪ MS_19200 ▪ MS_38400 ▪ MS_57600 ▪ MS_115200 • <UCPTsize>. The data bit size for messages sent over the Modbus network. A data bit is a group of 5 to 8 bits that represents a single character of data for transmission over the network. Data bits are preceded by a start bit, and they are followed by an optional parity bit and one or more stop bits. The default value is CS_8. This property may be one of the following values: ▪ CS_5 ▪ CS_6 ▪ CS_7 ▪ CS_8 • <UCPTparity>. The parity bit size for messages sent over the Modbus network. A parity bit is an extra bit used to check for errors in groups of data bits transferred between devices. The default parity size is P_NONE. This property may be one of the following values: ▪ P_NONE ▪ P_ODD ▪ P_EVEN • <UCPTstopBits>. The number of stop bits used on the Modbus network. The default value is SB_1. This property may be one of the following values: ▪ SB_1 ▪ SB_2 This property is not available for TCP/IP Modbus channels.

Property	Description
<UCPTserialMode>	The transmission mode used by the SmartServer for communicating with Modbus devices. The default transmission mode is MM_RTU. You have the following two choices: • MM_RTU. Data is sent as two 4-bit, hexadecimal characters. RTU mode provides a higher throughput than ASCII mode at equivalent baud rates. • MM_ASCII. Data is sent as two ASCII characters. ASCII mode provides increased flexibility in regards to the timing sequence, as there can be up to a 1-second interval between character transmissions without communication errors occurring.
<UCPTretryCount>	The number of times a network message is re-sent when no confirmation is received. The default value is 1 for Modbus channels.
<UCPTminOfflineTime>	If a network message fails, a data point and its device are marked offline. You can specify the <UCPTminOfflineTime> property so that all the data points on the offline device with pending network messages (read/write requests, polls, or heartbeats) are marked offline and network messages are not sent to them. This ensures that network performance is not impacted by an offline device. You can set the minimum period of time (in seconds) that the SmartServer waits before transmitting network messages to offline data points. During this period, an offline device transmits an OFFLINE status in response to data point requests. Once <UCPTminOfflineTime> elapses, the SmartServer sends a read/write request to one offline data point. If the read/write request succeeds, the data point and its device are marked online, and all cached read/write requests for the offline data points on the device are executed. This property is optional. If you do not specify this property in a *Set* function, the current value stored in it is erased. You must specify this property even if you are not changing it in order to preserve the current value. The default <UCPTminOfflineTime> for a Modbus channel is 60 seconds.

15.1.3 Using the Set Function on Modbus Channels

You can use the *Set* function to overwrite the configuration of a Modbus channel, or to create a new Modbus channel. The input parameters you supply to the function will include one or more <Item> elements. Each <Item> element includes a <UCPTname> property that specifies a unique Modbus channel to be created or modified.

Each <Item> element may also include a series of properties that define the configuration of the new (or modified) Modbus channel. This set of properties is the same whether you are creating a new Modbus channel or modifying an existing Modbus channel.

- If you are creating a new Modbus channel, you only need to specify the <UCPTchannelType> property; all other properties are optional.
- If you are modifying an existing Modbus channel, you must specify the channel's <UCPThandle>. If you do not specify the handle, a new channel is created. All other properties must be filled; otherwise the values stored in them are erased. The previous section, *Using the Get Function on Modbus Channels*, details the properties you can include in the *Set* function.

Request (create a new TCP/IP Modbus channel on the SmartServer)

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="MOD_Channel_Cfg">
        <UCPTname>Net/Modbus Channel 2</UCPTname>
        <UCPTannotation>IP</UCPTannotation>
        <UCPTchannelType LonFormat="UCPTchannelType">CT_TCP_IP_MASTER</UCPTchannelType>
    </Item>
  </iLonItem>
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="MOD_Channel_Cfg">
        <UCPTname>Net/Modbus Channel 2</UCPTname>
        <UCPTannotation>IP</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-04-02T11:43:37.460-07:00</UCPTlastUpdate>
        <UCPTuri>MOD_Channel_Cfg.htm</UCPTuri>
        <UCPThandle>1</UCPThandle>
        <UCPTchannelType LonFormat="UCPTchannelType">CT_TCP_IP_MASTER</UCPTchannelType>
        <UCPTport>502</UCPTport>
        <UCPTretryCount LonFormat="UCPTretryCount">1</UCPTretryCount>
    </Item>
  </iLonItem>
</SetResponse>
```

15.1.4 Using the Delete Function on Modbus Channels

You can use the *Delete* function to delete a Modbus channel on the SmartServer. The *Delete* function takes an <Item> element with a MOD_Channel_Cfg type as its input. The <Item> element only needs to include the Modbus channel's <UCPTname> property in the *Delete* Request as demonstrated in the following code sample:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="MOD_Channel_Cfg">
        <UCPTname>Net/Modbus Channel 2</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/Modbus Channel 2</UCPTname>
    </Item>
  </iLonItem>
```

15.2 Modbus Devices

The following section describes how to use the *List*, *Get*, *Set*, and *Delete* functions on Modbus devices.

15.2.1 Using the List Function on Modbus Devices

You can use the *List* function to retrieve a list of Modbus devices on the SmartServer. The *List* function takes an <iLonItem> element that has an xSelect statement with a MOD_Device_Cfg type as its input, as shown in the example below.

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="MOD_Device_Cfg"]</xSelect>
  </iLonItem>
</List>
```

Response

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
    <Item>
        <UCPTname>Net/Modbus Channel/LAE_LCD15_1</UCPTname>
        <UCPTannotation>App;xsi:type="MOD_Device_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
</iLonItem>
</List>
```

You could use the list of <Item> elements returned by this function as input for the *Get* function. The *Get* function would then return the configuration of each device included in the list. The next section describes the properties included in each of these elements.

15.2.2 Using the Get Function on Modbus Devices

You can use the *Get* function to retrieve the configuration of a Modbus device defined on the SmartServer. The input parameters you supply to this function will include one or more <Item> elements with a MOD_Device_Cfg type. Each <Item> element will include the <UCPTname> of each device whose configuration is to be returned by this function, as shown in the example below.

Request

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
 <iLonItem>
    <Item xsi:type="MOD_Device_Cfg">
        <UCPTname>Net/Modbus Channel/LAE_LCD15_1</UCPTname>
    </Item>
  </iLonItem>
</Get>
```

Response

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="MOD_Device_Cfg">
        <UCPTname>Net/Modbus Channel/LAE_LCD15_1</UCPTname>
        <UCPTannotation>App</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-04-02T12:15:22.960-07:00</UCPTlastUpdate>
        <UCPTuri>MOD_Device_Cfg.htm</UCPTuri>
        <UCPThandle>0</UCPThandle>
        <UCPTmaxElements>1</UCPTmaxElements>
        <Address>
            <UCPTaddress>0</UCPTaddress>
        </Address>
```

```
    </Item>
</iLonItem>
</Get>
```

The *Get* function returns an <Item> element for each device referenced in the input parameters you supplied to the function. The properties included within each <Item> element are initially defined when the device is added to the SmartServer. You can write to these device properties with the *Set* function. The following table describes these properties.

Property	Description
<UCPTname>	The name of the Modbus device in the following format: <*network*/*channel*/*device*>. You can rename a Modbus device by providing its <UCPThandle> and specifying the new <UCPTname> property to which the Modbus device is to be renamed.
<UCPTannotation>	The type of Modbus device (**App** by default) and its xsi type, which is MOD_Device_Cfg. This determines the icon used to represent the Modbus device in the SmartServer Web interface.
<UCPThidden>	A flag indicating whether the Modbus device is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **0** – shown **1** – hidden
<UCPTitemStatus>	• This property only appears if the device has the following exception: IS_OFFLINE. The device application is offline.
<UCPTlastUpdate>	A timestamp indicating the last time the configuration of the Modbus device was updated. This timestamp uses the ISO 8601 format, which is as follows: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Data Point was last updated. The second segment (after the **T):** HH:MM:SS.sss represents the time of day the configuration of the Data Point 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 be expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. It can be left out. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002. If it is not UTC, time shift has to be defined: 2008-02-28T09:59:53.660+01:00
<UCPTdescription>	A user-defined description of the Modbus device. This can be a maximum of 201 characters long.

Property	Description
<UCPTuri>	The name of the file on the SmartServer flash disk containing the configuration of the Modbus device. This property is always MOD_Device_Cfg.htm.
<UCPThandle>	The handle assigned to the Modbus device assigned by the SmartServer. When you use the *Set* function to modify the configuration of an existing Modbus device, you must specify the device's handle. If you do not specify the handle, a new Modbus device is created. You cannot use the *Set* function to modify the handle assigned to the Modbus device.
<UCPTmaxElements>	The maximum number of Modbus data points (registers) that can be transferred in one Modbus message.
<Address>	The logical address of the device on the Modbus network in decimal or hexadecimal format.

15.2.3 Using the Set Function on Modbus Devices

You can use the *Set* function to overwrite the configuration of a Modbus device, or to create a new Modbus device. The input parameters you supply to the function will include one or more <Item> elements. Each <Item> element includes a <UCPTname> property that specifies a unique Modbus device to be created or modified.

Each <Item> element may also include a series of properties that define the configuration of the new (or modified) Modbus device. This set of properties is the same whether you are creating a new Modbus device or modifying an existing Modbus device.

- If you are modifying an existing Modbus device, you must specify the <UCPThandle> property. In addition, all other properties should be filled; otherwise the values stored in them are erased. The previous section, *Using the Get Function on a Modbus Device*, details the properties you can include in the *Set* function.

You can set multiple Modbus devices with a single *Set* message. However, you should not attempt to create or write to more than 100 Modbus devices with a single call to the *Set* function. The following example demonstrates how to create a new Modbus device.

Request (create a new Modbus device on the SmartServer)

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
 <iLonItem>
    <Item xsi:type="MOD_Device_Cfg">
        <UCPTname>Net/Modbus Channel/LAE_LCD15_2</UCPTname>
        <UCPTmaxElements>1</UCPTmaxElements>
        <Address>
            <UCPTaddress>0</UCPTaddress>
        </Address>
    </Item>
  </iLonItem
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/Modbus Channel/LAE_LCD15_2</UCPTname>
    </Item>
  </iLonItem>
```

```
</SetResponse>
```

15.2.4 Using the Delete Function on Modbus Devices

You can use the *Delete* function to delete a Modbus device on the SmartServer. The *Delete* function takes an <Item> element with a MOD_Device_Cfg type as its input. The <Item> element only needs to include the device's <UCPTname> property in the *Delete* Request as demonstrated in the following code sample:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="MOD_Device_Cfg">
        <UCPTname>Net/Modbus Channel/LAE_LCD15_2</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/Modbus Channel/LAE_LCD15_2</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

15.3 Modbus Virtual Functional Blocks

Before you can add data points to a Modbus device, you need to use the *Set* function to create a Virtual functional block under the Modbus device. This virtual functional block is used to encapsulate the Modbus data points and enable the Modbus driver to adhere to the network hierarchy naming convention. The *Set* function takes an <Item> element with a MOD_Fb_Cfg type as its input, as shown in the example below.

Request (add a virtual functional block to a Modbus device)

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="MOD_Fb_Cfg">
        <UCPTname>Net/Modbus Channel/LAE_LCD15_2/VirtFb</UCPTname>
    </Item>
</iLonItem>
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/Modbus Channel/LAE_LCD15_2/VirtFb</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

15.4 Modbus Data Points

The following section describes how to use the *List*, *Get*, *Set*, and *Delete* functions on Modbus data points. For information on reading and writing values to Modbus data points, see Chapter 4, *Using the SmartServer Data Server*.

15.4.1 Using the List Function on Modbus Data Points

You can use the *List* function to retrieve a list of Modbus data points on the SmartServer. The List function takes an <iLonItem> element that has an xSelect statement with a MOD_Dp_Cfg type as its input, as shown in the example below.

Request (use an xSelect statement to return all the Modbus data points on the SmartServer)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="MOD_Dp_Cfg"]</xSelect>
  </iLonItem>
</List>
```

Alternatively, you can filter the Modbus data points returned by the *List* function to those on a specific device by including the <UCPTname>of the parent device in the xSelect statement, or you can filter the Modbus data points returned using the <UCPTname> and <UCPTlastUpdate> data point properties.

Request (use an xSelect statement to return all the Modbus data points on a specific device)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="MOD_Dp_Cfg"][starts-with(UCPTname,"Net/Modbus Channel/LAE_LCD15_2")]
    </xSelect>
  </iLonItem>
</List>
```

Request (use an xSelect statement to return all the Modbus data points that were updated after a specific time)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="MOD_Dp_Cfg"][UCPTlastUpdate&gt;"2008-03-31T00:00:00"]
    </xSelect>
  </iLonItem>
</List>
```

Request (return all the Modbus data points of a specific type based on name)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="MOD_Dp_Cfg"][contains(UCPTname,"Alrm")]</xSelect>
  </iLonItem>
</List>
```

Response

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <Item>
        <UCPTname>Net/Modbus Channel/LAE_LCD15_1/VirtFb/Alrm_1</UCPTname>
        <UCPTannotation>Dp_In_Out;xsi:type="MOD_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
    <Item>
        <UCPTname>Net/Modbus Channel/LAE_LCD15_1/VirtFb/Alrm-Prb1_1</UCPTname>
        <UCPTannotation>Dp_In_Out;xsi:type="MOD_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
    <Item>
        <UCPTname>Net/Modbus Channel/LAE_LCD15_1/VirtFb/Alrm-Prb2_1</UCPTname>
        <UCPTannotation>Dp_In_Out;xsi:type="MOD_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
    <Item>
        <UCPTname>Net/Modbus Channel/LAE_LCD15_1/VirtFb/Alrm-Prb3_1</UCPTname>
        <UCPTannotation>Dp_In_Out;xsi:type="MOD_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
</List>
```

15.4.2 Using the Get Function on Modbus Data Points

You can use the *Get* function to retrieve the configuration of Modbus data points defined on the SmartServer. The input parameters you supply to this function will include one or more <Item> elements with a MOD_Dp_Cfg type. Each <Item> element will include the <UCPTname> of each Modbus data point whose configuration is to be returned by this function, as shown in the example below.

Request (use an xSelect statement to return a specific Modbus data point)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="MOD_Dp_Cfg">
        <UCPTname>Net/Modbus Channel/LAE_LCD15_1/VirtFb/Alrm_1
    </Item>
  </iLonItem>
</Get>
```

Alternatively, you can specify one or more Modbus data point properties in the xSelect statement to filter the items returned by the *Get* function, including the <UCPTname> to filter data points based on their parent device.

Request (use an xSelect statement return all the Modbus data points on a specific device)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="MOD_Dp_Cfg"][starts-with(UCPTname,"Net/Modbus Channel/LAE_LCD15_1")]
    </xSelect>
 </iLonItem>
</Get>
```

Response

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="MOD_Dp_Cfg">
        <UCPTname>Net/Modbus Channel/LAE_LCD15_1/VirtFb/T1_1</UCPTname>
        <UCPTannotation>Dp_In_Out</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-04-02T12:15:22.020-07:00</UCPTlastUpdate>
        <UCPTdescription>Temperature 1</UCPTdescription>
        <UCPTuri>MOD_Dp_Cfg.htm</UCPTuri>
        <UCPTformatDescription>#8000010128000000[4].UNVT_signed_long#dec</UCPTformatDescription>
        <UCPTlength>2</UCPTlength>
        <UCPTunit field="">°C</UCPTunit>
        <UCPThandle>0</UCPThandle>
        <UCPTbaseType LonFormat="UCPTbaseType">BT_SIGNED_LONG</UCPTbaseType>
        <UCPTmodbusTable LonFormat="UCPTmodbusTable">MTT_HR</UCPTmodbusTable>
        <UCPTstartAddress>0</UCPTstartAddress>
        <UCPTstartBit>0</UCPTstartBit>
        <UCPTbitLength>16</UCPTbitLength>
        <UCPTdataOrdering LonFormat="UCPTdataOrdering">DO_BIG_ENDIAN</UCPTdataOrdering>
        <UCPTpollRate>20.0</UCPTpollRate>
    </Item>
  </iLonItem>
</Get>
```

The *Get* function returns an <Item> element for each Modbus data point referenced in the input parameters you supplied to the function. The properties included within each <Item> element are initially defined when the Modbus data point is added to the SmartServer. You can write to these data point properties with the *Set* function. The following table describes these properties.

Property	Description
<UCPTname>	The name of the Modbus data point in the following format: <*network*/*channel*/*device*/*functionalblock*/*data point*>. You can

Property	Description
	rename a Modbus data point by providing its <UCPThandle> and specifying the new <UCPTname> property to which the Modbus data point is to be renamed.
<UCPTannotation>	The type of Modbus data point, which is Dp_In_Out by default. This determines the icon used to represent the Modbus data point in the SmartServer Web interface.
<UCPThidden>	A flag indicating whether the Modbus data point is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **0** – shown **1** – hidden
<UCPTlastUpdate>	A timestamp indicating the last time the configuration of the Modbus data point was updated. This timestamp uses the ISO 8601 format, which is as follows: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Data Point was last updated. The second segment (after the **T):** HH:MM:SS.sss represents the time of day the configuration of the Data Point 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 be expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. It can be left out. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002. If it is not UTC, time shift has to be defined: 2008-02-28T09:59:53.660+01:00
<UCPTdescription>	A user-defined description of the Modbus data point. This can be a maximum of 201 characters long.
<UCPTuri>	The name of the file on the SmartServer flash disk containing the configuration of the Modbus data point. This property is MOD_Dp_Cfg.htm by default.
<UCPTformatDescription>	• The Modbus data point's program ID; data type (SNVT, SCPT, UNVT, UCPT, or built-in data type); and format (e.g., SI metric or US customary if the type has multiple formats. The format description is displayed in the following format: #<*manufacturer ID*>**[***scope selector***].**<*type name*>[#format] . This determines many factors about the Modbus data point,

Property	Description
	including the type of values it takes and its base type. If you do not set this property, it is set to RAW_HEX and the Modbus data point uses raw hex values.
<UCPTlength>	Specifies the size (in bytes) of the Modbus data point.
<UCPTunit>	This property is a string up to 227 characters long that describes the units the value in which a Modbus data point is measured. It is based on the data type assigned to the Modbus data point. A default value will be assigned to this property if a unit for the Modbus data point type chosen for the data point exists in the resource files on the SmartServer.
<UCPThandle>	The handle assigned to the Modbus data point assigned by the SmartServer. When you use the *Set* function to modify the configuration of an existing Modbus data point, you must specify the data point's handle. If you do not specify the handle, a new Modbus device is created. You cannot use the *Set* function to modify the handle assigned to the Modbus data point.
<UCPTbaseType>	The base type of a Modbus data point. This property may be one of the following values: • BT_DOUBLE • BT_FLOAT • BT_SIGNED_QUAD • BT_UNSIGNED_QUAD • BT_SIGNED_LONG • BT_UNSIGNED_LONG • BT_SIGNED_SHORT • BT_UNSIGNED_SHORT • BT_SIGNED_CHAR • BT_UNSIGNED_CHAR • BT_ENUM • BT_BITFIELD • BT_ARRAY • BT_STRUCT • BT_UNION
<UCPTmodbusTable>	The data access types based on the associated Modbus device This property may be one of the following values: • MTT_C [Coil Functions (Functions 1 & 5 single-write)]. For a single coil. Single bit, read-write data that has two

Property	Description
	states (on/off). • MTT_C_MO [Coil Functions (Functions 1 & 15 multi-write)]. For multiple coils. Single bit, read-write data that has two states (on/off). • MTT_DI [Discrete Input (Function 2)]. Single bit, read-only data that has two states (on/off). • MTT_IR [Input Register (Function 4)]. 16-bit read-only data that can be interpreted as a numeric value, a bit map, or an ASCII character. • MTT_HR [Hold Register (Functions 3 & 6 single-write)]. For a single register. 16-bit write data that can be interpreted as a numeric value, a bit map, or an ASCII character. • MTT_HR_M [Hold Register (Functions 3 & 16 multi-write)]. For multiple registers. 16-bit write data that can be interpreted as a numeric value, a bit map, or an ASCII character. This is the default.
<UCPTstartAddress>	The start address of the register to be used to read or write to the data point. If the <UCPTlength> property is configured to use bits, you can select the start and stop bits in the address. The Modbus driver is configured to ensure that the start and stop addresses remain consistent with the <UCPTlength> property. This means that if <UCPTlength> is changed, the <UCPTstartAddress> and <UCPTstartAddress> properties are automatically updated to fit the desired length. Similarly, if the <UCPTstartAddress> or <UCPTstartAddress> properties are changed, the <UCPTlength> property is updated accordingly.
<UCPTstopAddress>	The stop address of the register to be used to read or write to the data point.
<UCPTstartBit>	The start bit of a Modbus data word.
<UCPTbitLength>	The length of an Modbus data word in bits
<UCPTdataOrdering>	The ordering scheme to be used for interpreting Modbus data. This property may be one of the following values: • DO_BIG_ENDIAN. The highest order byte of data is sent first and all subsequent bytes of data are arranged from highest to lowest order. This is the default. • DO_LITTLE_ENDIAN. Lowest order byte of data is sent first, and each subsequent byte is arranged from lowest to highest order • DO_BYTE_SWAP. Data is first arranged from highest to lowest order, but every pair of bytes in the structure is interchanged.

Property	Description
	• DO_WORD_SWAP. Data is first arranged from highest to lowest order, but every pair of 16-bit words is swapped. For example, consider a device that uses an unsigned 32-bit integer to report runtime accumulation. Selecting the data ordering scheme is required because the Modbus protocol leaves the interpretation of 32-bit integers to the discretion of the implementer. In Big Endian format, the value of 120,000 hours (0x01D4C0 in hexadecimal format) would be represented as a value of: 00 01 D4 C0 in memory. This requires two adjacent Modbus registers (each holding 16 bits of data). If the device manufacture defines the unit runtime to be at register address 0x8, the Big Endian formatted response to a read function would return data 0x0001 0xD4C0 in that order. Now suppose the manufacture states the U32 value is returned in Little Endian format. One interpretation of the value returned to the driver from the read function would be 0xC0D4 0x0100. Alternatively, the manufacture may interpret Little Endian to be the ordering of registers and not bytes. In this case, the read function would return 0xD4C0 0x0001 and the driver would need to swap words to handle the value. If the device returned a value of 0x0100 0xC0D4A, a byte swapped format would need to be applied.
`<UCPTpollRate>`	The frequency in which the SmartServer's internal data server polls the Modbus data point. The recommended minimum poll rate is 30 seconds; the maximum poll rate is 1 second. The default poll rate for Modbus data points is **20** seconds. **Note**: The actual poll rate is determined by calculating the least common denominator of all the poll rates set for the data point from the applications to which it has been added.

15.4.3 Using the Set Function on Modbus Data Points

You can use the *Set* function to overwrite the configuration of a Modbus data point, or to create a new Modbus data point. The input parameters you supply to the function will include one or more <Item> elements. Each <Item> element includes a <UCPTname> property that specifies a unique Modbus data point to be created or modified.

Each <Item> element may also include a series of properties that define the configuration of the new (or modified) Modbus data point. This set of properties is the same whether you are creating a new Modbus data point or modifying an existing Modbus data point.

- If you are creating a new Modbus data point, you need to specify the <UCPTmodbusTable>, <UCPTstartAddress>, and <UCPTstartBit> properties; all other properties are optional.
- If you are modifying an existing Modbus data point, you must specify the <UCPThandle> property. In addition, all other properties should be filled; otherwise the values stored in them are erased. The previous section, *Using the Get Function on Modbus Data Points*, details the properties you can include in the *Set* function.

You can set multiple Modbus data points with a single *Set* message. However, you should not attempt to create or write to more than 100 Modbus data points with a single call to the *Set* function. The following example demonstrates how to create a new Modbus data point.

Request (add a Modbus data point to a Modbus device)

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
 <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="MOD_Dp_Cfg">
        <UCPTname>Net/Modbus Channel/LAE_LCD15_2/VirtFb/T1_1</UCPTname>
        <UCPTformatDescription>#8000010128000000[4].UNVT_signed_long#dec</UCPTformatDescription>
        <UCPTbaseType LonFormat="UCPTbaseType">BT_SIGNED_LONG</UCPTbaseType>
        <UCPTmodbusTable LonFormat="UCPTmodbusTable">MTT_HR</UCPTmodbusTable>
        <UCPTstartAddress>0</UCPTstartAddress>
        <UCPTstartBit>0</UCPTstartBit>
        <UCPTbitLength>16</UCPTbitLength>
    </Item>
  </iLonItem>
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/Modbus Channel/LAE_LCD15_2/VirtFb/T1_1</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

15.4.4 Using the Delete Function on Modbus Data Points

You can use the *Delete* function to delete a Modbus data point on the SmartServer. The *Delete* function takes an <Item> element with a MOD_Dp_Cfg type as its input. The <Item> element only needs to include the Modbus data point's <UCPTname> property in the *Delete* Request as demonstrated in the following code sample:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="MOD_Dp_Cfg">
        <UCPTname> Net/Modbus Channel/LAE_LCD15_2/VirtFb/T1_1</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/Modbus Channel/LAE_LCD15_2/VirtFb/T1_1</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

16 M-Bus Driver

The following chapter describes how to manage M-Bus channels, devices, and data points on the SmartServer.

16.1 M-Bus Channels

The following section describes how to use the *List*, *Get*, *Set*, and *Delete* functions on M-Bus channels.

16.1.1 Using the List Function on M-Bus Channels

You can use the *List* function to retrieve a list of M-Bus channels on the SmartServer. The List function takes an <iLonItem> element that has an xSelect statement with an MBS_Channel_Cfg type as its input, as shown in the example below.

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="MBS_Channel_Cfg"]</xSelect>
  </iLonItem>
</List>
```

Response

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
    <Item>
        <UCPTname>Net/M-Bus Channel</UCPTname>
        <UCPTannotation>RS232;xsi:type="MBS_Channel_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
  </iLonItem>
</List>
```

The *List* function returns a list of <Item> elements for each M-Bus channel defined on the SmartServer. You could use the list of <Item> elements returned by this function as input for the *Get* function. The *Get* function would then return the configuration of each M-Bus channel included in the list. The next section describes the properties included in each of these elements.

16.1.2 Using the Get Function on M-Bus Channels

You can use the *Get* function to retrieve the configuration of an M-Bus channel defined on the SmartServer. The input parameters you supply to this function will include one or more <Item> elements with an MBS_Channel_Cfg type. Each <Item> element will include the <UCPTname> of each channel whose configuration is to be returned by this function, as shown in the example below.

Request

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="MBS_Channel_Cfg">
        <UCPTname>Net/M-Bus Channel</UCPTname>
    </Item>
  </iLonItem>
</Get>
```

Response

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="MBS_Channel_Cfg">
        <UCPTname>Net/M-Bus Channel</UCPTname>
        <UCPTannotation>RS232</UCPTannotation>
```

```
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-04-02T13:06:24.430-07:00</UCPTlastUpdate>
        <UCPTuri>MBS_Channel_Cfg.htm</UCPTuri>
        <UCPThandle>0</UCPThandle>
        <UCPTchannelType LonFormat="UCPTchannelType">CT_RS232_MASTER</UCPTchannelType>
        <InterfaceOptions>
            <UCPTspeed LonFormat="UCPTspeed">MS_2400</UCPTspeed>
            <UCPTsize LonFormat="UCPTsize">CS_8</UCPTsize>
            <UCPTparity LonFormat="UCPTparity">P_EVEN</UCPTparity>
            <UCPTstopBits LonFormat="UCPTstopBits">SB_1</UCPTstopBits>
        </InterfaceOptions>
        <UCPTretryCount LonFormat="UCPTretryCount">1</UCPTretryCount>
    </Item>
  </iLonItem>
 </Get>
```

The *Get* function returns an <Item> element for each channel referenced in the input parameters you supplied to the function. The properties included within each <Item> element are initially defined when the channel is added to the SmartServer. You can write to these channel properties with the *Set* function. The following table describes these properties.

Property	Description
`<UCPTname>`	The name of the channel in the following format: <*network*/*channel*>. You can rename an M-Bus channel by providing its <UCPThandle> and specifying the new <UCPTname> property to which the channel is to be renamed.
`<UCPTannotation>`	The type of M-Bus channel, which may be RS232 or RS485. This determines the icon used to represent the M-Bus channel in the SmartServer Web interface.
`<UCPThidden>`	A flag indicating whether the M-Bus channel is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **`0`** – shown **`1`** – hidden
`<UCPTlastUpdate>`	A timestamp indicating the last time the configuration of the M-Bus channel was updated. This timestamp uses the ISO 8601 format, which is as follows: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Data Point was last updated. The second segment (after the **T):** HH:MM:SS.sss represents the time of day the configuration of the Data Point 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 be expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. It can be left out. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002.

Property	Description
	If it is not UTC, time shift has to be defined: 2008-02-28T09:59:53.660+01:00
<UCPTdescription>	A user-defined description of the channel. This can be a maximum of 201 characters long.
<UCPTuri>	The name of the file on the SmartServer flash disk containing the configuration of the M-Bus channel. This property is always MBS_Channel_Cfg.htm.
<UCPThandle>	The handle assigned to the M-Bus channel assigned by the SmartServer. When you use the *Set* function to modify the configuration of an existing M-Bus channel, you must specify the channel's handle. If you do not specify the handle, a new channel is created. You cannot use the *Set* function to modify the handle assigned to the channel.
<UCPTchannelType>	The channel type. This property may be one of the following values: • CT_RS485_MASTER. RS-485 is a balanced line, half-duplex system that allows transmission distances of up to 1.2 km. RS-485 allows for transmission over longer distances at higher speeds. This is the default. • CT_RS232_MASTER. RS-232 uses serial binary data for transmitting data between two devices.
<InterfaceOptions>	This element contains the following options for M-Bus channels: • <UCPTspeed>. The baud rate at which the SmartServer communicates with the M-Bus devices on the channel. The default value is MS_2400. This property may be one of the following values: ▪ MS_300 ▪ MS_600 ▪ MS_1200 ▪ MS_2400 ▪ MS_4800 ▪ MS_9600 • <UCPTsize>. The data bit size for messages sent over the M-Bus network. A data bit is a group of 5 to 8 bits that represents a single character of data for transmission over the network. Data bits are preceded by a start bit, and they are followed by an optional parity bit and one or more stop bits. The default value is CS_8. This property may be one of the following values: ▪ CS_5 ▪ CS_6 ▪ CS_7 ▪ CS_8 • <UCPTparity>. The parity bit size for messages sent over

Property	Description
	the M-Bus network. A parity bit is an extra bit used to check for errors in groups of data bits transferred between devices. The default parity size is P_EVEN. This property may be one of the following values: ▪ P_NONE ▪ P_ODD ▪ P_EVEN • <UCPTstopBits>. The number of stop bits used on the M-Bus network. The default value is SB_1. This property may be one of the following values: ▪ SB_1 ▪ SB_2 This property is not available for TCP/IP M-Bus channels.
`<UCPTretryCount>`	The number of times a network message is re-sent when no confirmation is received. The default value is 1 for M-Bus channels.
`<UCPTminOfflineTime>`	If a network message fails, a data point and its device are marked offline. You can specify the <UCPTminOfflineTime> property so that all the data points on the offline device with pending network messages (read/write requests, polls, or heartbeats) are marked offline and network messages are not sent to them. This ensures that network performance is not impacted by an offline device. You can set the minimum period of time (in seconds) that the SmartServer waits before transmitting network messages to offline data points. During this period, an offline device transmits an OFFLINE status in response to data point requests. Once <UCPTminOfflineTime> elapses, the SmartServer sends a read/write request to one offline data point. If the read/write request succeeds, the data point and its device are marked online, and all cached read/write requests for the offline data points on the device are executed. This property is optional. If you do not specify this property in a *Set* function, the current value stored in it is erased. You must specify this property even if you are not changing it in order to preserve the current value. The default <UCPTminOfflineTime> for an M-Bus channel is 60 seconds.

16.1.3 Using the Set Function on M-Bus Channels

You can use the *Set* function to overwrite the configuration of an M-Bus channel, or to create a new M-Bus channel. The input parameters you supply to the function will include one or more <Item> elements. Each <Item> element includes a <UCPTname> property that specifies a unique M-Bus channel to be created or modified.

Each <Item> element may also include a series of properties that define the configuration of the new (or modified) M-Bus channel. This set of properties is the same whether you are creating a new M-Bus channel or modifying an existing M-Bus channel.

- If you are creating a new M-Bus channel, you only need to specify the <UCPTchannelType> property; all other properties are optional.
- If you are modifying an existing channel, you must specify the channel's <UCPThandle>. If you do not specify the handle, a new channel is created. All other properties must be filled; otherwise the values stored in them are erased. The previous section, *Using the Get Function on M-Bus Channels*, details the properties you can include in the *Set* function.

Request (create a new M-Bus channel on the SmartServer)

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="MBS_Channel_Cfg">
        <UCPTname>Net/M-Bus Channel</UCPTname>
        <UCPTannotation>IP</UCPTannotation>
        <UCPTchannelType LonFormat="UCPTchannelType">CT_TCP_IP_MASTER</UCPTchannelType>
    </Item>
  </iLonItem>
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="MBS_Channel_Cfg">
        <UCPTname>Net/M-Bus Channel</UCPTname>
        <UCPTannotation>IP</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-04-02T11:43:37.460-07:00</UCPTlastUpdate>
        <UCPTuri>MBS_Channel_Cfg.htm</UCPTuri>
        <UCPThandle>1</UCPThandle>
        <UCPTchannelType LonFormat="UCPTchannelType">CT_TCP_IP_MASTER</UCPTchannelType>
        <UCPTport>502</UCPTport>
        <UCPTretryCount LonFormat="UCPTretryCount">1</UCPTretryCount>
    </Item>
  </iLonItem>
</SetResponse>
```

16.1.4 Using the Delete Function on M-Bus Channels

You can use the *Delete* function to delete an M-Bus channel on the SmartServer. The *Delete* function takes an <Item> element with an MBS_Channel_Cfg type as its input. The <Item> element only needs to include the M-Bus channel's <UCPTname> property in the *Delete* Request as demonstrated in the following code sample:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="MBS_Channel_Cfg">
        <UCPTname>Net/M-Bus Channel</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/M-Bus Channel</UCPTname>
    </Item>
  </iLonItem>
```

16.2 M-Bus Devices

The following section describes how to use the *List*, *Get*, *Set*, and *Delete* functions on M-Bus devices.

16.2.1 Using the List Function on M-Bus Devices

You can use the *List* function to retrieve a list of M-Bus devices on the SmartServer. The *List* function takes an <iLonItem> element that has an xSelect statement with an MBS_Device_Cfg type as its input, as shown in the example below.

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="MBS_Device_Cfg"]</xSelect>
  </iLonItem>
</List>
```

Response

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
    <Item>
        <UCPTname>Net/M-Bus Channel/M-Bus Device</UCPTname>
        <UCPTannotation>App;xsi:type="MBS_Device_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
</iLonItem>
</List>
```

You could use the list of <Item> elements returned by this function as input for the *Get* function. The *Get* function would then return the configuration of each device included in the list. The next section describes the properties included in each of these elements.

16.2.2 Using the Get Function on M-Bus Devices

You can use the *Get* function to retrieve the configuration of an M-Bus device defined on the SmartServer. The input parameters you supply to this function will include one or more <Item> elements with an MBS_Device_Cfg type. Each <Item> element will include the <UCPTname> of each device whose configuration is to be returned by this function, as shown in the example below.

Request

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
 <iLonItem>
    <Item xsi:type="MBS_Device_Cfg">
        <UCPTname>Net/M-Bus Channel/M-Bus Device</UCPTname>
    </Item>
  </iLonItem>
</Get>
```

Response

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="MBS_Device_Cfg">
        <UCPTname>Net/M-Bus Channel/M-Bus Device</UCPTname>
        <UCPTannotation>App</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-04-02T14:47:25.970-07:00</UCPTlastUpdate>
        <UCPTuri>MBS_Device_Cfg.htm</UCPTuri>
        <UCPThandle>0</UCPThandle>
        <UCPTspeed LonFormat="UCPTspeed">MS_2400</UCPTspeed>
        <UCPTmbusFcbEnable>1</UCPTmbusFcbEnable>
        <UCPTmbusFcvEnable>1</UCPTmbusFcvEnable>
        <UCPTmbusMode LonFormat="UCPTmbusMode">MD_MODE1</UCPTmbusMode>
```

```
        <UCPTmbusMedId>MED_NUL</UCPTmbusMedId>
        <UCPTmbusManId />
        <UCPTmbusGenId>ff</UCPTmbusGenId>
        <Address>
            <UCPTmbusAddressTyp LonFormat="UCPTmbusAddressTyp">AT_SECONDARY</UCPTmbusAddressTyp>
            <UCPTmbusPrimaryAddress>0</UCPTmbusPrimaryAddress>
            <UCPTmbusSecondaryAddress>00000000</UCPTmbusSecondaryAddress>
        </Address>
    </Item>
  </iLonItem>
</Get>
```

The *Get* function returns an <Item> element for each device referenced in the input parameters you supplied to the function. The properties included within each <Item> element are initially defined when the device is added to the SmartServer. You can write to these device properties with the *Set* function. The following table describes these properties.

Property	Description
<UCPTname>	The name of the M-Bus device in the following format: <*network*/*channel*/*device*>. You can rename an M-Bus device by providing its <UCPThandle> and specifying the new <UCPTname> property to which the M-Bus device is to be renamed.
<UCPTannotation>	The type of M-Bus device (**App** by default) and its xsi type, which is MBS_Device_Cfg. This determines the icon used to represent the M-Bus device in the SmartServer Web interface.
<UCPThidden>	A flag indicating whether the M-Bus device is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **0** – shown **1** – hidden
<UCPTitemStatus>	• This property only appears if the device has the following exception: IS_OFFLINE. The device application is offline.
<UCPTlastUpdate>	A timestamp indicating the last time the configuration of the M-Bus device was updated. This timestamp uses the ISO 8601 format, which is as follows: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Data Point was last updated. The second segment (after the **T):** HH:MM:SS.sss represents the time of day the configuration of the Data Point 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 be expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. It can be left out. For example, 2002-08-15T10:13:13Z indicates a UTC time of

Property	Description
	10:13:13 AM on August 15, 2002. If it is not UTC, time shift has to be defined: 2008-02-28T09:59:53.660+01:00
<UCPTdescription>	A user-defined description of the M-Bus device. This can be a maximum of 201 characters long.
<UCPTuri>	The name of the file on the SmartServer flash disk containing the configuration of the M-Bus device. This property is always MBS_Device_Cfg.htm.
<UCPThandle>	The handle assigned to the M-Bus device assigned by the SmartServer. When you use the *Set* function to modify the configuration of an existing M-Bus device, you must specify the device's handle. If you do not specify the handle, a new M-Bus device is created. You cannot use the *Set* function to modify the handle assigned to the M-Bus device.
<UCPTspeed>	• The baud rate (bits per second [bps)] at which the M-Bus device communicates on the serial port. See the documentation for your M-Bus device for more information on supported baud rates. The default value is MS_2400. This property may be one of the following values: • MS_300 • MS_600 • MS_1200 • MS_2400 • MS_4800 • MS_9600
<UCPTmbusFcbEnable>	Specifies whether the device uses the Frame Count-Bit (FCB) for sending requests. The default value is 1.
<UCPTmbusFcvEnable>	Specifies whether the device uses the Frame Count Valid Bit (FCV) for sending requests. The default value is 1. If <UCPTmbusFcbEnable> and <UCPTmbusFcvEnable> are set to 1, you can select the C field in the M-Bus request telegram: 0x4b fcv=0 fcb=0 0x5b fcv=1 fcb=0 0x6b fcv=0 fcb=1 0x7b fcv=1 fcb=1 **Multi-Telegram Support** If a total answer sequence from a device will not fit into a single RSP_UD telegram from the M-Bus device to the SmartServer, the last DIF is set to 0x1f. The SmartServer signals by a toggled FCB-Bit together with a set FCV-Bit in the next REQ_UD telegram that the last RSP_UD-telegram has been properly

Property	Description
	received from the device. The device answers to a REQ_UD-request with toggled FCB-Bit with a set FCV-bit from the SmartServer with a RSP_UD containing the data telegram section of a multi-telegram answer. **Notes**: An M-Bus device with a single RSP_UD-telegram may ignore the FCB in the REQ_UD2-telegram and always send the same (single) telegram. An M-Bus device with exactly two (sequential) RSP_UD-answer telegrams may use the FCB of the REQ_UD2 to decide which of both telegrams should be transmitted.
`<UCPTmbusMode>`	The communication mode for the requested device. This property may be one of the following values: • MD_MODE1 • MD_MODE2
`<UCPTmbusMedId>`	The device's medium ID as a 1-byte enumeration that identifies the device functionality. This read-only property is filled when the device responds to a request correctly. The following is a list of possible medium IDs: MED_NUL — Invalid MED_OTH ER — Others MED_OIL — Oil MED_ELECTRICITY — Electricity MED_GAS — Gas MED_RETURN_TEMP — Return temperature MED_STEAM — Steam MED_HOT_WATER — Hot water MED_WATER — Water MED_HEAT_METER — Heat meter MED_COMPRESSED_AIR — Compressed-air MED_RES1 — Reserved MED_RES2 — Reserved MED_FLOW_TEMP — Flow temperature, outgoing/supply temperature MED_RES3 — Reserved MED_SYS_BUS — System / Bus MED_UNKNOWN — Unknown
`<UCPTmbusManId>`	The device's manufacturer ID as a 3-byte string. This read-only property is filled when the device responds to a request correctly.
`<UCPTmbusGenId>`	The generation or version of the device as 1-byte char. This read-only property is filled when the device responds to a request correctly. The value depends on the manufacturer.
`<Address>`	This element contains the following options for M-Bus devices: • <UCPTmbusAddressTyp>. Specifies whether the M-Bus device uses primary (AT_PRIMARY) or secondary

Property	Description
	(AT_SECONDARY) addressing. Primary addressing is preferred because it makes replacing M-Bus devices more transparent. Each of these addressing methods is described as follows: ▪ **Primary**. The primary address is assigned by the network management tool used to install the M-Bus device (analogous to a LONWORKS subnet/icon address). ▪ **Secondary**. The secondary address is burned into the device at the factory (analogous to a LONWORKS Neuron ID). • <UCPTmbusPrimaryAddress>. The primary address must be between 0 to 250. • <UCPTmbusSecondaryAddress>. The secondary address must be between 0 to 99,999,999.

16.2.3 Using the Set Function on M-Bus Devices

You can use the *Set* function to overwrite the configuration of an M-Bus device, or to create a new M-Bus device. The input parameters you supply to the function will include one or more <Item> elements. Each <Item> element includes a <UCPTname> property that specifies a unique M-Bus device to be created or modified.

Each <Item> element may also include a series of properties that define the configuration of the new (or modified) M-Bus device. This set of properties is the same whether you are creating a new M-Bus device or modifying an existing M-Bus device.

- If you are modifying an existing M-Bus device, you must specify the <UCPThandle> property. In addition, all other properties should be filled; otherwise the values stored in them are erased. The previous section, *Using the Get Function on M-Bus Devices*, details the properties you can include in the *Set* function.

You can set multiple M-Bus devices with a single *Set* message. However, you should not attempt to create or write to more than 100 M-Bus devices with a single call to the *Set* function. The following example demonstrates how to create a new M-Bus device.

Request (create a new M-Bus device on the SmartServer)

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
 <iLonItem>
    <Item xsi:type="MBS_Device_Cfg">
        <UCPTname>Net/M-Bus Channel/M-Bus Device 2</UCPTname>
    </Item>
  </iLonItem
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/M-Bus Channel/M-Bus Device 2</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

16.2.4 Using the Delete Function on M-Bus Devices

You can use the *Delete* function to delete an M-Bus device on the SmartServer. The *Delete* function takes an <Item> element with an MBS_Device_Cfg type as its input. The <Item> element only needs to include the device's <UCPTname> property in the *Delete* Request as demonstrated in the following code sample:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="MBS_Device_Cfg">
        <UCPTname>Net/M-Bus Channel/M-Bus Device 2</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/M-Bus Channel/M-Bus Device 2</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

16.3 M-Bus Virtual Functional Blocks

Before you can add data points to an M-Bus device, you need to use the *Set* function to create a Virtual functional block under the M-Bus device. This virtual functional block is used to encapsulate the M-Bus data points and enable the M-Bus driver to adhere to the network hierarchy naming convention. The *Set* function takes an <Item> element with an MBS_Fb_Cfg type as its input, as shown in the example below.

Request (add a virtual functional block to an M-Bus **device)**

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="MBS_Fb_Cfg">
        <UCPTname>Net/M-Bus Channel/M-Bus Device/VirtFb</UCPTname>
    </Item>
</iLonItem>
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/M-Bus Channel/M-Bus Device/VirtFb</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

16.4 M-Bus Data Points

The following section describes how to use the *List*, *Get*, *Set*, and *Delete* functions on M-Bus data points. For information on reading and writing values to M-Bus data points, see Chapter 4, *Using the SmartServer Data Server*.

16.4.1 Using the List Function on M-Bus Data Points

You can use the *List* function to retrieve a list of M-Bus data points on the SmartServer. The List function takes an <iLonItem> element that has an xSelect statement with an MBS_Dp_Cfg type as its input, as shown in the example below.

Request (return all the M-Bus data points on the SmartServer)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="MBS_Dp_Cfg"]</xSelect>
  </iLonItem>
</List>
```

Alternatively, you can filter the M-Bus data points returned by the *List* function to those on a specific device by including the <UCPTname>of the parent device in the xSelect statement, or you can filter the M-Bus data points returned using the <UCPTname> and <UCPTlastUpdate> data point properties.

Request (use an xSelect statement to return all the M-Bus data points on a specific device)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="MBS_Dp_Cfg"][starts-with(UCPTname,"Net/M-Bus Channel/M-Bus Device")]
    </xSelect>
  </iLonItem>
</List>
```

Request (use an xSelect statement to return all the M-Bus data points that were updated after a specific time)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="MBS_Dp_Cfg"][UCPTlastUpdate&gt;"2008-03-31T00:00:00"]
    </xSelect>
  </iLonItem>
</List>
```

Request (return all the M-Bus data points of a specific type based on name)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="MBS_Dp_Cfg"][contains(UCPTname,"In")]</xSelect>
  </iLonItem>
</List>
```

Response

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
       <Item>
       <UCPTname>Net/M-Bus Channel/M-Bus Device/Virtual Fb/In</UCPTname>
       <UCPTannotation>Dp_In;xsi:type="MBS_Dp_Cfg"</UCPTannotation>
       <UCPThidden>0</UCPThidden>
   </Item>
</List>
```

16.4.2 Using the Get Function on M-Bus Data Points

You can use the *Get* function to retrieve the configuration of M-Bus data points defined on the SmartServer. The input parameters you supply to this function will include one or more <Item> elements with an MBS_Dp_Cfg type. Each <Item> element will include the <UCPTname> of each Modbus data point whose configuration is to be returned by this function, as shown in the example below.

Request (use an xSelect statement to return a specific M-Bus data point)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="MBS_Dp_Cfg">
        <UCPTname>Net/M-Bus Channel/M-Bus Device/Virtual Fb/In</UCPTname>
    </Item>
```

```
  </iLonItem>
</Get>
```

Alternatively, you can specify one or more M-Bus data point properties in the xSelect statement to filter the items returned by the *Get* function, including the <UCPTname> to filter data points based on their parent device.

Request (use an xSelect statement return all the M-Bus data points on a specific device)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="MBS_Dp_Cfg"][starts-with(UCPTname,"Net/M-Bus Channel/LAE_LCD15_1")]
    </xSelect>
 </iLonItem>
</Get>
```

Response

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
        <Item xsi:type="MBS_Dp_Cfg">
        <UCPTname>Net/M-Bus Channel/M-Bus Device/Virtual Fb/In</UCPTname>
        <UCPTannotation>Dp_In</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-04-02T15:29:01.570-07:00</UCPTlastUpdate>
        <UCPTuri>MBS_Dp_Cfg.htm</UCPTuri>
        <UCPTformatDescription>RAW_HEX</UCPTformatDescription>
        <UCPTlength>1</UCPTlength>
        <UCPTdirection LonFormat="UCPTdirection">DIR_IN</UCPTdirection>
        <UCPThandle>0</UCPThandle>
        <UCPTpollRate>10.0</UCPTpollRate>
        <UCPTmbusType LonFormat="UCPTmbusType">MBST_READ</UCPTmbusType>
    </Item>
  </iLonItem>
</Get>
```

The *Get* function returns an <Item> element for each M-Bus data point referenced in the input parameters you supplied to the function. The properties included within each <Item> element are initially defined when the M-Bus data point is added to the SmartServer. You can write to these data point properties with the *Set* function. The following table describes these properties.

Property	Description
<UCPTname>	The name of the M-Bus data point in the following format: <*network*/*channel*/*device*/*functionalblock*/*data point*>. You can rename an M-Bus data point by providing its <UCPThandle> and specifying the new <UCPTname> property to which the M-Bus data point is to be renamed.
<UCPTannotation>	The type of M-Bus data point, which is Dp_In by default. This determines the icon used to represent the M-Bus data point in the SmartServer Web interface.
<UCPThidden>	A flag indicating whether the M-Bus data point is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **0** – shown **1** – hidden
<UCPTlastUpdate>	A timestamp indicating the last time the configuration of the M-Bus data point was updated. This timestamp uses the ISO 8601 format, which is as follows:

Property	Description
	YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Data Point was last updated. The second segment (after the **T):** HH:MM:SS.sss represents the time of day the configuration of the Data Point 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 be expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. It can be left out. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002. If it is not UTC, time shift has to be defined: 2008-02-28T09:59:53.660+01:00
`<UCPTdescription>`	A user-defined description of the M-Bus data point. This can be a maximum of 201 characters long.
`<UCPTuri>`	The name of the file on the SmartServer flash disk containing the configuration of the M-Bus data point. This property is MBS_Dp_Cfg.htm by default.
`<UCPTformatDescription>`	• The M-Bus data point's program ID; data type (SNVT, SCPT, UNVT, UCPT, or built-in data type); and format (e.g., SI metric or US customary if the type has multiple formats. The format description is displayed in the following format: #<*manufacturer ID*>**[***scope selector***].**<*type name*>[#format] . This determines many factors about the M-Bus data point, including the type of values it takes and its base type. If you do not set this property, it is set to RAW_HEX and the M-Bus data point uses raw hex values.
`<UCPTlength>`	Specifies the size (in bytes) of the M-Bus data point.
`<UCPTdirection>`	Specifies whether the Modbus data point is an input data point (DIR_IN), output data point (DIR_OUT), or undefined (DIR_NUL).
`<UCPThandle>`	The handle assigned to the M-Bus data point assigned by the SmartServer. When you use the *Set* function to modify the configuration of an existing M-Bus data point, you must specify the data point's handle. If you do not specify the handle, a new M-Bus device is created. You cannot use the *Set* function to modify the handle assigned to the M-Bus data point.

Property	Description
<UCPTpollRate>	The frequency in which the SmartServer's internal data server polls the M-Bus data point. The recommended minimum poll rate is 30 seconds; the maximum poll rate is 1 second. The default poll rate for M-Bus data points is **10** seconds. **Note**: The actual poll rate is determined by calculating the least common denominator of all the poll rates set for the data point from the applications to which it has been added.
<UCPTmbusType>	Specifies the type (read or write) of the M-Bus data point. This property may have one of the following values: • MBST_READ • MBST_WRITE

16.4.3 Using the Set Function on M-Bus Data Points

You can use the *Set* function to overwrite the configuration of an M-Bus data point, or to create a new M-Bus data point. The input parameters you supply to the function will include one or more <Item> elements. Each <Item> element includes a <UCPTname> property that specifies a unique M-Bus data point to be created or modified.

Each <Item> element may also include a series of properties that define the configuration of the new (or modified) M-Bus data point. This set of properties is the same whether you are creating a new M-Bus data point or modifying an existing M-Bus data point.

- If you are modifying an existing M-Bus data point, you must specify the <UCPThandle> property. In addition, all other properties should be filled; otherwise the values stored in them are erased. The previous section, *Using the Get Function on M-Bus Data Points*, details the properties you can include in the *Set* function.

You can set multiple M-Bus data points with a single *Set* message. However, you should not attempt to create or write to more than 100 M-Bus data points with a single call to the *Set* function. The following example demonstrates how to create a new M-Bus data point.

Request (add an M-Bus data point to an M-Bus device)

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="MBS_Dp_Cfg">
        <UCPTname>Net/M-Bus Channel/M-Bus Device/VirtFb/MBS3</UCPTname>
        <UCPTannotation>Dp_In</UCPTannotation>
        <UCPTformatDescription>#8000014600000000[4].UCPT_MBS3</UCPTformatDescription>
        <UCPTdirection xsi:type="string" LonFormat="UCPTdirection">DIR_IN</UCPTdirection>
        <UCPTlength>24</UCPTlength>
        <UCPTpollRate>10.0</UCPTpollRate>
     </Item>
  </iLonItem>
<Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/M-Bus Channel/M-Bus Device/VirtFb/MBS3</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

16.4.4 Using the Delete Function on M-Bus Data Points

You can use the *Delete* function to delete an M-Bus data point on the SmartServer. The *Delete* function takes an <Item> element with an MBS_Dp_Cfg type as its input. The <Item> element only needs to include the M-Bus data point's <UCPTname> property in the *Delete* Request as demonstrated in the following code sample:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="MBS_Dp_Cfg">
        <UCPTname>Net/M-Bus Channel/M-Bus Device/VirtFb/MBS3</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/M-Bus Channel/M-Bus Device/VirtFb/MBS3</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

17 Virtual Driver

The virtual channel is the SmartServer's internal channel. It is used as a gateway for system information that is used by the data points on the SmartServer. The Virtual driver contains data points representing the SmartServer's free RAM, free disk space, CPU usage, software version number, last received service pin message, and other information. The following chapter describes how to manage virtual channels, and devices and data points on the virtual channel.

17.1 Virtual Channels

The following section describes how to use the *List*, *Get*, *Set*, and *Delete* functions on Virtual channels.

17.1.1 Using the List Function on Virtual Channels

You can use the *List* function to retrieve a list of Virtual channels on the SmartServer. The List function takes an <iLonItem> element that has an xSelect statement with a Virtual_Channel_Cfg type as its input, as shown in the example below.

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="Virtual_Channel_Cfg"]</xSelect>
  </iLonItem>
</List>
```

Response

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
    <Item>
        <UCPTname>Net/VirtCh</UCPTname>
        <UCPTannotation>VirtualChannel;xsi:type="Virtual_Channel_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
  </iLonItem>
</List>
```

The *List* function returns a list of <Item> elements for each Virtual channel defined on the SmartServer. You could use the list of <Item> elements returned by this function as input for the *Get* function. The *Get* function would then return the configuration of each Virtual channel included in the list. The next section describes the properties included in each of these elements.

17.1.2 Using the Get Function on Virtual Channels

You can use the *Get* function to retrieve the configuration of a Virtual channel defined on the SmartServer. The input parameters you supply to this function will include one or more <Item> elements with a Virtual_Channel_Cfg type. Each <Item> element will include the <UCPTname> of each channel whose configuration is to be returned by this function, as shown in the example below.

Request

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="Virtual_Channel_Cfg">
        <UCPTname>Net/Virtual Channel</UCPTname>
    </Item>
  </iLonItem>
</Get>
```

Response

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
```

```
  <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="Virtual_Channel_Cfg">
        <UCPTname>Net/Virtual Channel</UCPTname>
        <UCPTannotation>VirtualChannel</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-04-02T17:05:45.270-07:00</UCPTlastUpdate>
        <UCPTuri>Virtual_Channel_Cfg.htm</UCPTuri>
        <UCPThandle>1</UCPThandle>
        <UCPTchannelType>Virtual</UCPTchannelType>
    </Item>
  </iLonItem>
 </Get>
```

The *Get* function returns an <Item> element for each channel referenced in the input parameters you supplied to the function. The properties included within each <Item> element are initially defined when the channel is added to the SmartServer. You can write to these channel properties with the *Set* function. The following table describes these properties.

Property	Description
`<UCPTname>`	The name of the channel in the following format: <*network*/*channel*>. You can rename a Virtual channel by providing its <UCPThandle> and specifying the new <UCPTname> property to which the channel is to be renamed.
`<UCPTannotation>`	The type of Virtual channel, which is Virtual Channel. This determines the icon used to represent the Virtual channel in the SmartServer Web interface.
`<UCPThidden>`	A flag indicating whether the Virtual channel is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **`0`** – shown **`1`** – hidden
`<UCPTlastUpdate>`	A timestamp indicating the last time the configuration of the Virtual channel was updated. This timestamp uses the ISO 8601 format, which is as follows: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Data Point was last updated. The second segment (after the **T):** HH:MM:SS.sss represents the time of day the configuration of the Data Point 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 be expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. It can be left out. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002. If it is not UTC, time shift has to be defined: 2008-02-28T09:59:53.660+01:00

Property	Description
<UCPTdescription>	A user-defined description of the channel. This can be a maximum of 201 characters long.
<UCPTuri>	The name of the file on the SmartServer flash disk containing the configuration of the Virtual channel. This property is always Virtual_Channel_Cfg.htm.
<UCPThandle>	The handle assigned to the Virtual channel assigned by the SmartServer. When you use the *Set* function to modify the configuration of an existing Virtual channel, you must specify the channel's handle. If you do not specify the handle, a new channel is created. You cannot use the *Set* function to modify the handle assigned to the channel.
<UCPTchannelType>	• The channel type, which is always Virtual.

17.1.3 Using the Set Function on Virtual Channels

You can use the *Set* function to overwrite the configuration of a Virtual channel, or to create a new Virtual channel. The input parameters you supply to the function will include one or more <Item> elements. Each <Item> element includes a <UCPTname> property that specifies a unique Virtual channel to be created or modified.

Each <Item> element may also include a series of properties that define the configuration of the new (or modified) Virtual channel. This set of properties is the same whether you are creating a new Virtual channel or modifying an existing Virtual channel.

- If you are creating a new Virtual channel, you only need to specify the <UCPTchannelType> property and set it to "Virtual"; all other properties are optional.
- If you are modifying an existing channel, you must specify the channel's <UCPThandle>. If you do not specify the handle, a new channel is created. All other properties must be filled; otherwise the values stored in them are erased. The previous section, *Using the Get Function on Virtual Channels*, details the properties you can include in the *Set* function. You can create up to two Virtual channels with a single *Set* message.

Request (create a new Virtual channel on the SmartServer)

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="Virtual_Channel_Cfg">
        <UCPTname>Net/Virtual Channel 1</UCPTname>
        <UCPTannotation>VirtualChannel</UCPTannotation>
        <UCPTchannelType>Virtual</UCPTchannelType>
    </Item>
  </iLonItem>
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="Virtual_Channel_Cfg">
        <UCPTname>Net/Virtual Channel 1</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

17.1.4 Using the Delete Function on a Virtual Channel

You can use the *Delete* function to delete a Virtual channel on the SmartServer. The *Delete* function takes an <Item> element with a Virtual_Channel_Cfg type as its input. The <Item> element only needs to include the Virtual channel's <UCPTname> property in the *Delete* Request as demonstrated in the following code sample:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="Virtual_Channel_Cfg">
        <UCPTname>Net/Virtual Channel 1</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/Virtual Channel 1</UCPTname>
    </Item>
  </iLonItem>
```

17.2 Virtual Devices

The following section describes how to use the *List*, *Get*, *Set*, and *Delete* functions on Virtual devices.

17.2.1 Using the List Function on Virtual Devices

You can use the *List* function to retrieve a list of Virtual devices on the SmartServer. The *List* function takes an <iLonItem> element that has an xSelect statement with a Virtual_Device_Cfg type as its input, as shown in the example below.

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="Virtual_Device_Cfg"]</xSelect>
  </iLonItem>
</List>
```

Response

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
    <Item>
        <UCPTname>Net/VirtCh/iLON System</UCPTname>
        <UCPTannotation>VirtualDevice;local;xsi:type="Virtual_Device_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
</iLonItem>
</List>
```

You could use the list of <Item> elements returned by this function as input for the *Get* function. The *Get* function would then return the configuration of each device included in the list. The next section describes the properties included in each of these elements.

17.2.2 Using the Get Function on Virtual Devices

You can use the *Get* function to retrieve the configuration of a Virtual device defined on the SmartServer. The input parameters you supply to this function will include one or more <Item>

elements with a Virtual_Device_Cfg type. Each <Item> element will include the <UCPTname> of each device whose configuration is to be returned by this function, as shown in the example below.

Request

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
 <iLonItem>
    <Item xsi:type="Virtual_Device_Cfg">
        <UCPTname>Net/VirtCh/iLON System</UCPTname>
    </Item>
  </iLonItem>
</Get>
```

Response

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="Virtual_Device_Cfg">
        <UCPTname>Net/VirtCh/iLON System</UCPTname>
        <UCPTannotation>VirtualDevice;local</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-26T11:09:27.420-07:00</UCPTlastUpdate>
        <UCPTuri>Virtual_Device_Cfg.htm</UCPTuri>
        <UCPThandle>0</UCPThandle>
        <UCPTprogramId>0000000000000000</UCPTprogramId>
    </Item>
  </iLonItem>
</Get>
```

The *Get* function returns an <Item> element for each device referenced in the input parameters you supplied to the function. The properties included within each <Item> element are initially defined when the device is added to the SmartServer. You can write to these device properties with the *Set* function. The following table describes these properties.

Property	Description
<UCPTname>	The name of the Virtual device in the following format: <*network*/*channel*/*device*>. You can rename a Virtual device by providing its <UCPThandle> and specifying the new <UCPTname> property to which the Virtual device is to be renamed.
<UCPTannotation>	The type of device, which is "Virtual Device", and its location relative to the SmartServer, which is " local ". This determines the icon used to represent the Virtual device in the SmartServer Web interface.
<UCPThidden>	A flag indicating whether the Virtual device is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **0** – shown **1** – hidden
<UCPTlastUpdate>	A timestamp indicating the last time the configuration of the Virtual device was updated. This timestamp uses the ISO 8601 format, which is as follows: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Data Point was last updated. The second segment (after the **T):** HH:MM:SS.sss

Property	Description
	represents the time of day the configuration of the Data Point 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 be expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. It can be left out. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002. If it is not UTC, time shift has to be defined: 2008-02-28T09:59:53.660+01:00
`<UCPTdescription>`	A user-defined description of the Virtual device. This can be a maximum of 201 characters long.
`<UCPTuri>`	The name of the file on the SmartServer flash disk containing the configuration of the Virtual device. This property is always Virtual_Device_Cfg.htm.
`<UCPThandle>`	The handle assigned to the Virtual device assigned by the SmartServer. When you use the *Set* function to modify the configuration of an existing Virtual device, you must specify the device's handle. If you do not specify the handle, a new Virtual device is created. You cannot use the *Set* function to modify the handle assigned to the Virtual device.
`<UCPTprogramId>`	The program ID of the Virtual device as a set of hex digits.

17.2.3 Using the Set Function on Virtual Devices

You can use the *Set* function to overwrite the configuration of a Virtual device, or to create a new Virtual device. The input parameters you supply to the function will include one or more <Item> elements. Each <Item> element includes a <UCPTname> property that specifies a unique Virtual device to be created or modified.

Each <Item> element may also include a series of properties that define the configuration of the new (or modified) Virtual device. This set of properties is the same whether you are creating a new Virtual device or modifying an existing Virtual device.

- If you are modifying an existing Virtual device, you must specify the <UCPThandle> property. In addition, all other properties should be filled; otherwise the values stored in them are erased. The previous section, *Using the Get Function on Virtual Devices*, details the properties you can include in the *Set* function.

You can set multiple Virtual devices with a single *Set* message. However, you should not attempt to create or write to more than 100 Virtual devices with a single call to the *Set* function. The following example demonstrates how to create a new Virtual device.

Request (create a new Virtual device on the SmartServer)

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
 <iLonItem>
    <Item xsi:type="Virtual_Device_Cfg">
```

```
        <UCPTname>Net/Virtual Channel/Virtual Device</UCPTname>
    </Item>
  </iLonItem
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/Virtual Channel/Virtual Device</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

17.2.4 Using the Delete Function on Virtual Devices

You can use the *Delete* function to delete a Virtual device on the SmartServer. The *Delete* function takes an <Item> element with a Virtual_Device_Cfg type as its input. The <Item> element only needs to include the device's <UCPTname> property in the *Delete* Request as demonstrated in the following code sample:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="Virtual_Device_Cfg">
        <UCPTname>Net/Virtual Channel 1/Virtual Device</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/Virtual Channel 1/Virtual Device</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

17.3 Virtual Functional Blocks

Before you can add data points to a Virtual device, you need to use the *Set* function to create a virtual functional block under the Virtual device. This virtual functional block is used to encapsulate the virtual data points and enable the Virtual driver to adhere to the network hierarchy naming convention. The *Set* function takes an <Item> element with a Virtual_Fb_Cfg type as its input, as shown in the example below.

Request (add a virtual functional block to a Virtual device)

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="Virtual_Fb_Cfg">
        <UCPTname>Net/Virtual Channel 1/Virtual Device/VirtFb</UCPTname>
    </Item>
</iLonItem>
</Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/Virtual Channel 1/Virtual Device/VirtFb</UCPTname>
    </Item>
```

```
  </iLonItem>
</SetResponse>
```

17.4 Virtual Data Points

The following section describes how to use the *List*, *Get*, *Set*, and *Delete* functions on Virtual data points. For information on reading and writing values to Virtual data points, see Chapter 4, *Using the SmartServer Data Server*.

17.4.1 Using the List Function on Virtual Data Points

You can use the *List* function to retrieve a list of Virtual data points on the SmartServer. The List function takes an <iLonItem> element that has an xSelect statement with a Virtual_Dp_Cfg type as its input, as shown in the example below.

Request (return all the Virtual data points on the SmartServer)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="Virtual_Dp_Cfg"]</xSelect>
  </iLonItem>
</List>
```

Alternatively, you can filter the Virtual data points returned by the *List* function to those on a specific device by including the <UCPTname>of the parent device in the xSelect statement, or you can filter the Virtual data points returned using the <UCPTname> and <UCPTlastUpdate> data point properties.

Request (use an xSelect statement to return all the Virtual data points on a specific device)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="Virtual_Dp_Cfg"][starts-with(UCPTname,"Net/Virtual Channel/Virtual
    Device")]
    </xSelect>
  </iLonItem>
</List>
```

Request (use an xSelect statement to return all the Virtual data points that were updated after a specific time)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="Virtual_Dp_Cfg"][UCPTlastUpdate&gt;"2008-03-31T00:00:00"]
    </xSelect>
  </iLonItem>
</List>
```

Request (return all the Virtual data points related to the connection manager based on name)

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="Virtual_Dp_Cfg"][contains(UCPTname,"CM")]</xSelect>
  </iLonItem>
</List>
```

Response

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <UCPTcurrentConfig>4.0</UCPTcurrentConfig>
    <Item>
        <UCPTname>Net/VirtCh/iLON System/VirtFb/CMdialInNum</UCPTname>
        <UCPTannotation>Dp_Out;xsi:type="Virtual_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
    <Item>
        <UCPTname>Net/VirtCh/iLON System/VirtFb/CMgprsIp</UCPTname>
        <UCPTannotation>Dp_Out;xsi:type="Virtual_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
```

```
    </Item>
    <Item>
        <UCPTname>Net/VirtCh/iLON System/VirtFb/CMdialInIp</UCPTname>
        <UCPTannotation>Dp_Out;xsi:type="Virtual_Dp_Cfg"</UCPTannotation>
        <UCPThidden>0</UCPThidden>
    </Item>
  </iLonItem>
</List>
```

17.4.2 Using the Get Function on Virtual Data Points

You can use the *Get* function to retrieve the configuration of Virtual data points defined on the SmartServer. The input parameters you supply to this function will include one or more <Item> elements with a Virtual_Dp_Cfg type. Each <Item> element will include the <UCPTname> of each Modbus data point whose configuration is to be returned by this function, as shown in the example below.

Request (use an xSelect statement to return a specific Virtual data point)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="Virtual_Dp_Cfg">
        <UCPTname>Net/VirtCh/iLON System/VirtFb/CMdialInNum</UCPTname>
    </Item>
  </iLonItem>
</Get>
```

Alternatively, you can specify one or more Virtual data point properties in the xSelect statement to filter the items returned by the *Get* function, including the <UCPTname> to filter data points based on their parent device.

Request (use an xSelect statement return all the Virtual data points on a specific device)

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
        <xSelect>//Item[@xsi:type="Virtual_Dp_Cfg"][starts-with(UCPTname,"Net/VirtCh/iLON
        System/VirtFb")]
    </xSelect>
 </iLonItem>
</Get>
```

Response

```
<Get xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="Virtual_Dp_Cfg">
        <UCPTname>Net/VirtCh/iLON System/VirtFb/CMdialInNum</UCPTname>
        <UCPTannotation>Dp_Out</UCPTannotation>
        <UCPThidden>0</UCPThidden>
        <UCPTlastUpdate>2008-03-26T11:09:27.420-07:00</UCPTlastUpdate>
        <UCPTuri>Virtual_Dp_Cfg.htm</UCPTuri>
        <UCPTformatDescription>#0000000000000000[0].SNVT_str_asc</UCPTformatDescription>
        <UCPTlength>31</UCPTlength>
        <UCPTdirection LonFormat="UCPTdirection">DIR_OUT</UCPTdirection>
        <UCPThandle>0</UCPThandle>
    </Item>
  </iLonItem>
</Get>
```

The *Get* function returns an <Item> element for each Virtual data point referenced in the input parameters you supplied to the function. The properties included within each <Item> element are initially defined when the Virtual data point is added to the SmartServer. You can write to these data point properties with the *Set* function. The following table describes these properties.

Property	Description
<UCPTname>	The name of the Virtual data point in the following format:

Property	Description
	<*network*/*channel*/*device*/*functionalblock*/*data point*>. You can rename a Virtual data point by providing its <UCPThandle> and specifying the new <UCPTname> property to which the Virtual data point is to be renamed.
`<UCPTannotation>`	The type of Virtual data point, which may be Dp_In, Dp_Out, or Dp_In_Out (undefined). This determines the icon used to represent the Virtual data point in the SmartServer Web interface.
`<UCPThidden>`	A flag indicating whether the Virtual data point is hidden or shown in the navigation pane on the left side of the SmartServer Web interface. This property may have the following values: **`0`** – shown **`1`** – hidden
`<UCPTlastUpdate>`	A timestamp indicating the last time the configuration of the Virtual data point was updated. This timestamp uses the ISO 8601 format, which is as follows: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm The first segment of the time stamp (YYYY-MM-DD) represents the date the configuration of the Data Point was last updated. The second segment (after the **T):** HH:MM:SS.sss represents the time of day the configuration of the Data Point 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 be expressed as 16:00 UTC. The **Z** appended to the timestamp indicates that it is in UTC. It can be left out. For example, 2002-08-15T10:13:13Z indicates a UTC time of 10:13:13 AM on August 15, 2002. If it is not UTC, time shift has to be defined: 2008-02-28T09:59:53.660+01:00
`<UCPTdescription>`	A user-defined description of the Virtual data point. This can be a maximum of 201 characters long.
`<UCPTuri>`	The name of the file on the SmartServer flash disk containing the configuration of the Virtual data point. This property is Virtual_Dp_Cfg.htm by default.
`<UCPTformatDescription>`	• The Virtual data point's program ID; data type (SNVT, SCPT, UNVT, UCPT, or built-in data type); and format (e.g., SI metric or US customary if the type has multiple formats. The format description is displayed in the following format:

Property	Description
	#<*manufacturer ID*>**[***scope selector***].**<*type name*>[#format] . This determines many factors about the Virtual data point, including the type of values it takes and its base type. If you do not set this property, it is set to RAW_HEX and the Virtual data point uses raw hex values.
<UCPTlength>	Specifies the size (in bytes) of the Virtual data point.
<UCPTdirection>	Specifies whether the Modbus data point is an input data point (DIR_IN), output data point (DIR_OUT), or undefined (DIR_NUL).
<UCPThandle>	The handle assigned to the Virtual data point assigned by the SmartServer. When you use the *Set* function to modify the configuration of an existing Virtual data point, you must specify the data point's handle. If you do not specify the handle, a new Virtual device is created. You cannot use the *Set* function to modify the handle assigned to the Virtual data point.
<UCPTpollRate>	The frequency in which the SmartServer's internal data server polls the Virtual data point. The recommended minimum poll rate is 30 seconds; the maximum poll rate is 1 second. Polling for Virtual data points is disabled (0 seconds) by default. **Note**: The actual poll rate is determined by calculating the least common denominator of all the poll rates set for the data point from the applications to which it has been added.

17.4.3 Using the Set Function on Virtual Data Points

You can use the *Set* function to overwrite the configuration of a Virtual data point, or to create a new Virtual data point. The input parameters you supply to the function will include one or more <Item> elements. Each <Item> element includes a <UCPTname> property that specifies a unique Virtual data point to be created or modified.

Each <Item> element may also include a series of properties that define the configuration of the new (or modified) Virtual data point. This set of properties is the same whether you are creating a new Virtual data point or modifying an existing Virtual data point.

- If you are creating a new Virtual data point, you should specify the <UCPTformatDescription> property; all other properties are optional.
- If you are modifying an existing Virtual data point, you must specify the <UCPThandle> property. In addition, all other properties should be filled; otherwise the values stored in them are erased. The previous section, *Using the Get Function on Virtual Data Points*, details the properties you can include in the *Set* function.

You can set multiple Virtual data points with a single *Set* message. However, you should not attempt to create or write to more than 100 Virtual data points with a single call to the *Set* function. The following example demonstrates how to create a new Virtual data point.

Request (add a Virtual data point to a Virtual device)

```
<Set xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="Virtual_Dp_Cfg">
        <UCPTname>Net/VirtCh/iLON System/VirtFb/virt_switch</UCPTname>
```

```
        <UCPTannotation>Dp_Out</UCPTannotation>
        <UCPTformatDescription>#0000000000000000[0].SNVT_switch</UCPTformatDescription>
        <UCPTdirection LonFormat="UCPTdirection">DIR_OUT</UCPTdirection>
        <UCPTpollRate>1.0</UCPTpollRate>
    </Item>
  </iLonItem>
<Set>
```

Response

```
<SetResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/VirtCh/iLON System/VirtFb/virt_switch</UCPTname>
    </Item>
  </iLonItem>
</SetResponse>
```

17.4.4 Using the Delete Function on Virtual Data Points

You can use the *Delete* function to delete a Virtual data point on the SmartServer. The *Delete* function takes an <Item> element with a Virtual_Dp_Cfg type as its input. The <Item> element only needs to include the Virtual data point's <UCPTname> property in the *Delete* Request as demonstrated in the following code sample:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="Virtual_Dp_Cfg">
        <UCPTname>Net/VirtCh/iLON System/VirtFb/virt_switch</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<DeleteResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem >
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item>
        <UCPTname>Net/VirtCh/iLON System/VirtFb/virt_switch</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

18 File System Data

You can use the *List*, *Read*, *Write*, and *Delete* functions to download, upload, and delete the data logs, alarm logs, the eventlog.txt file on the root directory of the SmartServer flash disk, and other user-defined .txt and .csv files under the web/user directory on the SmartServer flash disk.

18.1 Using the List Function on File System Data

You can use the *List* function to return a list of directories containing .txt or .csv files on the SmartServer. The *List* function takes an <iLonItem> element that has an xSelect statement with a FileSystem_Data type as its input, as shown in the example below.

Request

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//Item[@xsi:type="FileSystem_Data"]</xSelect>
  </iLonItem>
</List>
```

Response

```
<List xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <UCPTfaultCount>0</UCPTfaultCount>
    <Item xsi:type="Item_Service">
        <UCPTname>/alarmLog/</UCPTname>
        <UCPTlastUpdate>2007-12-20T01:03:38Z</UCPTlastUpdate>
        <UCPTdescription />
        <UCPTfileSize>2048</UCPTfileSize>
    </Item>
    <Item xsi:type="Item_Service">
        <UCPTname>/data/</UCPTname>
        <UCPTlastUpdate>2007-12-20T01:03:40Z</UCPTlastUpdate>
        <UCPTdescription />
        <UCPTfileSize>2048</UCPTfileSize>
    </Item>
    <Item xsi:type="Item_Service">
        <UCPTname>/web/</UCPTname>
        <UCPTlastUpdate>2007-12-20T01:03:40Z</UCPTlastUpdate>
        <UCPTdescription />
        <UCPTfileSize>2048</UCPTfileSize>
    </Item>
    <Item xsi:type="Item_Service">
        <UCPTname>/eventlog.txt</UCPTname>
        <UCPTlastUpdate>2008-02-05T17:32:46Z</UCPTlastUpdate>
        <UCPTdescription />
        <UCPTfileSize>25000</UCPTfileSize>
    </Item>
  </iLonItem>
</List>
```

18.2 Using the Read Function on File System Data

You can use the *Read* function to download the contents of a specific alarm log, data log, the event log file on the root directory on the SmartServer flash disk, or other .txt or .csv file in the web/user directory on the SmartServer flash disk.

The *Read* function takes an <iLonItem> element that has an xSelect statement with a FileSystem type as its input, as shown in the example below. You can have the data returned in text format by specifying the <UCPTfileFormat> attribute and setting it to “text”. Otherwise, the data will be returned as a hex dump.

Request (return the data in the event log file on the SmartServer)

```
<Read xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
```

```
    <xSelect>//*[@xsi:type="FileSystem"][UCPTname="/eventlog.txt"][UCPTfileFormat="text"]</xSelect>
  </iLonItem>
</Read>
```

Request (return the data in a Data Logger)

```
<Read xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <xSelect>//*[@xsi:type="FileSystem"][UCPTname="/root/data/Net/LON/iLON App/Data
    Logger[1].csv"][UCPTfileFormat="text"]</xSelect>
  </iLonItem>
</Read>
```

Response

```
<Read xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="FileSystem_Data">
        <UCPTname>/root/data/Net/LON/iLON App/Data Logger[1].csv</UCPTname>
        <UCPTlastUpdate>2008-02-27T21:41:56Z</UCPTlastUpdate>
        <UCPTdescription />
        <UCPTfileFormat LonFormat="UCPTfileFormat">text</UCPTfileFormat>
        <UCPTfileSize>98052</UCPTfileSize>
        <Value>
        "UCPTlogTime","UCPTpointName","UCPTaliasName","Reserved","UCPTpointStatus","UCPTvalueDef","UC
        PTvalue","UCPTunit","UCPTpriority"

        2008-02-27T13:41:57.440-08:00,"Net/LON/iLON App/Digital Input
        2/nvoClsValue_2","NVL_nvoClsValue_2","","AL_NO_CONDITION","OFF","0.0 0","","255"

        2008-02-27T13:45:00.000-08:00,"Net/LON/iLON App/Digital Input
        2/nvoClsValue_2","NVL_nvoClsValue_2","","AL_NO_CONDITION","OFF","0.0 0","","255"

        2008-02-27T14:00:00.040-08:00,"Net/LON/iLON App/Digital Input
        2/nvoClsValue_2","NVL_nvoClsValue_2","","AL_NO_CONDITION","OFF","0.0 0","","255"
        </Value>
  </iLonItem>
<Read>
```

The following C# sample code demonstrates how to read a file in hex mode:

```
ilonWebRef.Item_Coll itemColl = new ilonWebRef.Item_Coll();

itemColl.xSelect =
"//*[@xsi:type=\"FileSystem\"][UCPTname=\"/root/data/myfile.txt\"][UCPTfileFormat=\"hex\"]";

ilonWebRef.Item_DataColl readResp = ilon.Read(itemColl);

if (readResp.UCPTfaultCount > 0)
    MessageBox.Show(readResp.fault.faultstring);
else
    String str = (readResp.Item[0] as ilonWebRef.FileSystem_Data).Value;
```

Note: You can read and write file system data that includes new line characters. For example, using the *Read* function on the following file system data:

```
<Read xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <iLonItem>
        <Item xsi:type="FileSystem_Data">
            <UCPTname>/data/writetest.txt</UCPTname>
            <UCPTfileFormat>text</UCPTfileFormat>
                    <Value>
                            Hello
                            World
                            !
                    </Value>
        </Item>
    </iLonItem>
<Read>
```

returns the following data:

```
Hello
World
!
```

18.3 Using the Write Function on File System Data

You can use the *Write* function to upload a new alarm log, data log, or other user-defined file to the SmartServer or to overwrite an existing file.

The *Write* function takes an <Item> element with a FileSystem_Data type as its input. You can specify the format (text or hex) of the file in the <UCPTfileFormat> property. If you do not include the <UCPTfileFormat> property and set it to "text", the data is stored as a hex dump. The data to be stored in the file must be included within the <Value> property.

The *Write* response message includes a <UCPTfileSize> that contains the length of the new file in bytes.

Request (create a new user-defined file on the SmartServer)

```
<Write xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="FileSystem_Data">
        <UCPTname>/data/myCustomFile.csv</UCPTname>
        <UCPTfileFormat>text</UCPTfileFormat>
        <Value>Hello i.LON SmartServer</Value>
    </Item>
  </iLonItem>
</Write>
```

Response

```
<Write xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="FileSystem_Data">
        <UCPTname>/Data/Test</UCPTname>
        <UCPTlastUpdate>2008-04-04T00:00:24Z</UCPTlastUpdate>
        <UCPTdescription />
        <UCPTfileFormat LonFormat="UCPTfileFormat">text</UCPTfileFormat>
        <UCPTfileSize>31</UCPTfileSize>
        <Value>
          Hello i.LON SmartServer
        </Value>
    </Item>
 </iLonItem>
<Write>
```

Note: You can write to a file in hex mode, and store the data in ASCII printable characters (text) and control characters (e.g., new line). To do this, you enter a space-separated ascii hex string in the <Value> property in the *Write* request. For example, you can write two lines to a text file (the first is "ABC" and the second is "DEF") by passing in the following ascii hex string in the <Value> property:

```
<Value>41 42 43 0D 0A 44 45 46</Value>
```

The following C# sample code demonstrates how to write to a file in hex mode using the previous example:

```
ilonWebRef.Item_DataColl itemDataColl = new ilonWebRef.Item_DataColl();
itemDataColl.Item = new ilonWebRef.Item_Data[1];
ilonWebRef.FileSystem_Data fsData = new ilonWebRef.FileSystem_Data();
itemDataColl.Item[0] = fsData;

fsData.UCPTname = "/data/myfile.txt";
fsData.UCPTfileFormat = new ilonWebRef.E_LonString();
fsData.UCPTfileFormat.Value = "hex";

fsData.Value = "41 42 43 0D 0A 44 45 46";
```

```
ilonWebRef.Item_Coll wrteResp = ilon.Write(itemDataColl);
```

18.4 Using the Delete Function on File System Data

You can use the *Delete* function to delete an alarm log, data log, or other user-defined file on the SmartServer. The *Delete* function takes an <Item> element with a FileSystem_Data type as its input. The <Item> element only needs to include the file's <UCPTname> property in the *Delete* request as demonstrated in the following code sample:

Request

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="FileSystem_Data">
        <UCPTname>/Data/Test</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

Response

```
<Delete xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLonItem>
    <Item xsi:type="FileSystem_Data">
        <UCPTname>/Data/Test</UCPTname>
    </Item>
  </iLonItem>
</Delete>
```

19 System Information Methods

You can use the *SystemService_Read_Info* and *SystemService_Write_Info* functions in the **iLON100_System.wsdl** to read system information about the SmartServer and modify the SmartServer's configuration. You can use the *SystemService_Test* function to test connections to various host devices such as SMTP mail servers, SNTP time servers, and remote SmartServers and read the results of the tests.

Note: To use the system information methods, you must reference the **iLON100_System.wsdl** in your code. See *Chapter 20, Using the SOAP Interface as a Web Service*, for more information on adding service references.

19.1 System Service Methods

You can use the *SystemService_Read_Info* and *SystemService_Write_Info* functions to read system information about the SmartServer and modify the SmartServer's configuration.

The *SystemService_Read_Info* and *SystemService_Write_Info* functions take a string consisting of an <iLONSystemService> element that includes one <UCPTsystemInfoType> property. The <UCPTsystemInfoType> property specifies an enumeration representing the type of system information to be read from and written to the SmartServer. The following example demonstrates how to use the *SystemService_Read_Info* function in your code:

```
string systemData = "<iLONSystemService><UCPTsystemInfoType>SI_TIME
</UCPTsystemInfoType></iLONSystemService>"
```

Tip: Section 21.1.7, *Configuring the SmartServer in Visual C# .NET*, includes a C# programming example demonstrating how to use the *SystemService_Read_Info* and *SystemService_Write_Info* functions. Section 21.2.7, *Configuring the SmartServer in Visual Basic.NET*, includes a Visual Basic example demonstrating how to do this.

The following table lists the values to which you can set <UCPTsystemInfoType> and the type of system information returned by the *SystemService_Read_Info* function:

<UCPTsystemInfoType>	**Data Returned by SystemService_Read_Info**
SI_NETWORK	SmartServer IP connections, including IPv4, IPv6, and DNS server connections.
SI_TIME	SmartServer real-time clock settings.
SI_SECURITY	SmartServer security settings.
SI_STATIC (read-only)	General information about the SmartServer, including the following: • Hardware and firmware versions. • Flash memory capacity and limits. • FPM license status.
SI_REAL_TIME (read-only)	General network performance statistics for the SmartServer, including the following: • Current flash memory usage. • CPU utilization. • Data point message failures.

	• Time of last refresh.
SI_MAIL	E-mail (SMTP) server settings.
SI_RTR	IP-852 router settings.
SI_RTR_STAT (read-only)	Network performance statistics for the SmartServer as an IP-852 router.
SI_LSPA	LonScanner Protocol Analyzer settings.

The *SystemService_Read_Info* function returns a string listing all the system information provided by the <UCPTsystemInfoType> property specified in the request message. You can use the *SystemService_Write_Info* function to configure the read/write properties of the item. The tables that follow the code samples for the various items list whether a property is writable (r/w) or read-only (r). Note that the system module also includes a simple *SystemService_Reboot* function that you can use to reboot a SmartServer.

19.1.1 TCP/IP Settings

You can use the *SystemService_Read_Info* function to get the SmartServer's IP connections, including its IPv4, IPv6, and DNS server connections. To do this, you provide the *SystemService_Read_Info* function with an <iLONSystemService> element that includes one <UCPTsystemInfoType> property that is set to SI_NETWORK.

Request

```
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <UCPTsystemInfoType>SI_NETWORK</UCPTsystemInfoType>
  </iLONSystemService>
</SystemService_Read_Info>
```

Response

```
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <NETWORK>
        <UCPTipAddress>10.2.124.82</UCPTipAddress>
        <UCPTipMask>255.255.128.0</UCPTipMask>
        <UCPTgateway>10.2.0.1</UCPTgateway>
        <UCPTdnsServer1>0.0.0.0</UCPTdnsServer1>
        <UCPTdnsServer2>0.0.0.0</UCPTdnsServer2>
        <UCPTipv6DnsServer1 />
        <UCPTipv6DnsServer2 />
        <UCPThostName>SmartServer</UCPThostName>
        <UCPTdomainSuffix />
        <UCPTdhcpEnabled>0</UCPTdhcpEnabled>
        <UCPTdnsDomainViaDhcpEnabled>0</UCPTdnsDomainViaDhcpEnabled>
        <UCPTdnsAddressViaDhcpEnabled>0</UCPTdnsAddressViaDhcpEnabled>
        <UCPTsecureAccessAllowed>1</UCPTsecureAccessAllowed>
        <UCPTmacID>00-D0-71-02-0A-18</UCPTmacID>
        <UCPTcurrentIpAddress>10.2.124.82</UCPTcurrentIpAddress>
        <UCPTcurrentIpMask>255.255.128.0</UCPTcurrentIpMask>
        <UCPTcurrentGateway>10.2.0.1</UCPTcurrentGateway>
        <UCPTgprsIpAddress />
        <UCPTipv4Only>1</UCPTipv4Only>
        <UCPTipv6StaticConfigEnable>0</UCPTipv6StaticConfigEnable>
        <UCPTipv6AddressStatic>::/64</UCPTipv6AddressStatic>
        <UCPTipv6Gateway />
        <UCPTipv6AddressAutoConfigLocal>fe80::2d0:71ff:fe02:a18</UCPTipv6AddressAutoConfigLocal>
        <UCPTipv6AddressAutoConfigGlobal />
        <UCPTipv6AddressViaDHCPEnable>0</UCPTipv6AddressViaDHCPEnable>
        <UCPTipv6DNSServerViaDHCPEnable>0</UCPTipv6DNSServerViaDHCPEnable>
        <UCPTipv6DNSDomainViaDHCPv6Enabled>0</UCPTipv6DNSDomainViaDHCPv6Enabled>
        <UCPTipv6AddressDHCP>TBD</UCPTipv6AddressDHCP>
```

```
        <UCPTethernetMode>Auto</UCPTethernetMode>
        <UCPTSoftwareDisabledByDownRevHardware>false</UCPTSoftwareDisabledByDownRevHardware>
    </NETWORK>
  </iLONSystemService>
</SystemService_Read_Info>
```

The *SystemService_Read_Info* function returns one <NETWORK> item. The following table lists and describes the properties of the <NETWORK> item. You can use the *SystemService_Write_Info* function to configure those properties that are marked (r/w). Properties that are read-only are marked with an (r).

Property	**Description**	**R/W**
`<UCPTipAddress>`	The SmartServer's configured IPv4 address. You must reboot the SmartServer to implement changes made to this property.	r/w
`<UCPTipMask>`	The SmartServer's configured IPv4 mask. You must reboot the SmartServer to implement changes made to this property.	r/w
`<UCPTgateway>`	The SmartServer's configured IPv4 gateway. You must reboot the SmartServer to implement changes made to this property.	r/w
`<UCPTdnsServer1>`	The SmartServer's first IPv4 DNS server	r/w
`<UCPTdnsServer2>`	The SmartServer's second IPv4 DNS server (fail over)	r/w
`<UCPTipv6dnsServer1>`	The SmartServer's first IPv6 DNS server	r/w
`<UCPTipv6dnsServer2>`	The SmartServer's second IPv6 DNS server (fail over)	r/w
`<UCPThostName>`	The SmartServer's hostname, which is "SmartServer" by default.	r/w
`<UCPTdomainSuffix>`	The SmartServer's domain suffix	r/w
`<UCPTdhcpEnabled>`	A flag indicating whether IPv4 DHCP is enabled.	r/w
`<UCPTdnsDomainViaDhcpEnabled>`	A flag indicating whether acquiring the DNS domain via DHCP is enabled.	r/w
`<UCPTdnsAddressViaDhcpEnabled>`	A flag indicating whether acquiring the DNS server address via DHCP enabled.	r/w
`<UCPTsecureAccessAllowed>`	A flag indicating whether secure access mode is enabled.	r/w
`<UCPTmacID>`	The MAC identification number used for the SmartServer's Ethernet port	r
`<UCPTcurrentIpAddress>`	The SmartServer's current configured IPv4 address. This property does not reflect changes made to the IPv4	r

Property	Description	R/W
	address prior to a reboot.	
<UCPTcurrentIpMask>	The SmartServer's current configured IPv4 mask. This property does not reflect changes made to the IPv4 mask prior to a reboot.	r
<UCPTcurrentGateway>	The SmartServer's current configured IPv4 gateway. This property does not reflect changes made to the IPv4 gateway prior to a reboot.	r
<UCPTgprsIpAddress>	If the SmartServer connects to the LAN via a GPRS modem connection, this is the SmartServer's IP address assigned by the provider	r
<UCPTipv4Only>	A flag indicating whether the IP mode used is IPV4 only (1) or IPV4/IPV6 (0).	r/w
<UCPTipv6StaticConfigEnable>	A flag indicating whether IPV6 static configuration is enabled.	r/w
<UCPTipv6AddressStatic>	The SmartServer's static IPV6 address	r/w
<UCPTipv6Gateway>	The SmartServer's static IPV6 gateway address	r/w
<UCPTipv6AddressAutoConfigLocal>	The SmartServer's link local IPV6 address	r
<UCPTipv6AddressAutoConfigGlobal>	The SmartServer's router advisement IPV6 address	r
<UCPTipv6AddressViaDHCPEnable>	A flag indicating whether acquiring the SmartServer's IPv6 address via DHCP is enabled.	r/w
<UCPTipv6DNSServerViaDHCPEnable>	A flag indicating whether acquiring the SmartServer's IPv6 DNS server via DHCP is enabled.	r/w
<UCPTipv6DNSDomainViaDHCPv6Enabled>	A flag indicating whether acquiring the SmartServer's IPv6 DNS server domain via DHCP is enabled.	r/w
<UCPTipv6AddressDHCP>	The SmartServer's IPv6 address assigned by the DHCP server.	r

19.1.2 Time Settings

You can use the *SystemService_Read_Info* function to get the SmartServer's real-time clock settings. To do this, you provide the *SystemService_Read_Info* function with an <iLONSystemService> element that includes one <UCPTsystemInfoType> property that is set to SI_TIME.

Request

```
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <UCPTsystemInfoType>SI_TIME</UCPTsystemInfoType>
```

```
  </iLONSystemService>
</SystemService_Read_Info>
```

Response

```
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <TIME>
        <UCPTtimeServer1>10.1.0.21</UCPTtimeServer1>
        <UCPTtimeServer2>192.6.38.127</UCPTtimeServer2>
        <UCPTtimeLastSynched>FRI APR 04 12:57:39 2008</UCPTtimeLastSynched>
        <UCPTsystemTime>2008-04-04T12:57:41</UCPTsystemTime>
        <UCPTtimeZone>(GMT-0800) Pacific:-480:1:2.1.3.2:1.1.11.2</UCPTtimeZone>
    </TIME>
  </iLONSystemService>
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
```

The *SystemService_Read_Info* function returns one <TIME> item. The following table lists and describes the properties of the <TIME> item. You can use the *SystemService_Write_Info* function to configure those properties that are marked (r/w). Properties that are read-only are marked (r).

Property	Description	R/W
<UCPTtimeServer1>	The IP address of the designated default SNTP time server.	r/w
<UCPTtimeServer2>	The IP address of the designated backup SNTP time server.	r/w
<UCPTtimeLastSynched>	The last time in which the SmartServer synchronized its clock with the default SNTP time server. The amount of time varies between 1 to 15 minutes, depending on the difference in time between the SmartServer's clock and the SNTP time server. As the difference approaches 75 ms or less, the interval will keep increasing until it reaches the maximum of 15 minutes.	r
<UCPTsystemTime>	Displays the time and date currently stored in the SmartServer's real time clock. **Note**: If you have configured an SNTP time server, changes to the time and date will be overwritten the next time the SmartServer is synchronized with the SNTP time server.	r
<UCPTtimeZone>	The time zone in which the SmartServer is located.	r/w

19.1.3 Security Settings

You can use the *SystemService_Read_Info* function to get the SmartServer's security settings. To do this, you provide the *SystemService_Read_Info* function with an <iLONSystemService> element that includes one <UCPTsystemInfoType> property that is set to SI_SECURITY.

Request

```
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <UCPTsystemInfoType>SI_SECURITY</UCPTsystemInfoType>
  </iLONSystemService>
</SystemService_Read_Info>
```

Response

```
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
        <SECURITY>
        <UCPTauthKeyRaw>00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00</UCPTauthKeyRaw>
        <UCPTauthKeyHashed />
        <UCPTrtrAuthKeyRaw>Not Available</UCPTrtrAuthKeyRaw>
        <UCPTautoAnswer>1</UCPTautoAnswer>
        <UCPTtelnetEnable>1</UCPTtelnetEnable>
        <UCPTftpEnable>1</UCPTftpEnable>
        <UCPTrniEnable>1</UCPTrniEnable>
        <UCPTftpUserName>ilon</UCPTftpUserName>
        <UCPTftpPassword />
        <UCPTremoteBootEnable>1</UCPTremoteBootEnable>
        <UCPTsecureAccessAllowed>1</UCPTsecureAccessAllowed>
        <UCPTsecureAccessAlways>1</UCPTsecureAccessAlways>
        <UCPThttpPort>80</UCPThttpPort>
        <UCPTftpPort>21</UCPTftpPort>
        <UCPTtelnetPort>23</UCPTtelnetPort>
        <UCPTwebEnable>1</UCPTwebEnable>
        <UCPTsysReboot />
        <UCPTsysFactoryDefault />
    </SECURITY>
  </iLONSystemService>
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
```

The *SystemService_Read_Info* function returns one <SECURITY> item. The following table lists and describes the properties of the <TIME> item. You can use the *SystemService_Write_Info* function to configure those properties that are marked (r/w). Properties that are read-only are marked (r).

Property	**Description**	**R/W**
<UCPTauthKeyRaw>	The MD5 authentication key used for authentication when using the SmartServer as an RNI. This value must match the one specified in the LONWORKS Interfaces control panel application.	r/w
<UCPTauthKeyHashed>	A text secret phase for authentication when using the SmartServer as an RNI. This can be used instead of the Raw MD5 authentication key specified by <UCPTauthKeyRaw>.	r/w
<UCPTautoAnswer>	A flag indicating whether the modem is responding to incoming calls.	r
<UCPTtelnetEnable	A flag indicating whether the SmartServer console application can be accessed via Telnet.	r/w
<UCPTftpEnable>	A flag indicating whether the SmartServer can be accessed via FTP.	r/w
<UCPTrniEnable>	A flag indicating whether the SmartServer can be used as an RNI.	r/w
<UCPTftpUserName>	The user name used to access the SmartServer via FTP. The default user name is ilon. You can specify a different user name, which may be up to 20 characters long and contain letters, numerals, and the underscore character.	r/w

Property	Description	R/W
<UCPTftpPassword	The password used to access the SmartServer via FTP. The default password is ilon. You can specify a different password, which may be up to 20 characters long and contain letters, numerals, and the underscore character.	r/w
<UCPTremoteBootEnable>	A flag indicating whether the SmartServer can be rebooted remotely via the **Setup - Reboot** Web page.	r/w
<UCPTsecureAccessAllowed>	A flag indicating whether the security settings on the **Setup – Security** Web page can be modified if secure access mode is temporarily disabled.	r/w
<UCPTsecureAccessAlways>	A flag indicating whether the security settings on the **Setup – Security** Web page can be modified if secure access mode is permanently disabled.	r/w
<UCPThttpPort>	The port the SmartServer uses for HTTP communication. The default port is 80.	r/w
<UCPThttpsPort>	The port the SmartServer uses for HTTPS communication. The default port is 443.	r/w
<UCPTftpPort>	The port the SmartServer uses for HTTP communication. The default port is 80.	r/w
<UCPTtelnetPort>	The port the SmartServer uses for HTTP communication. The default port is 80.	r/w
<UCPTwebEnable>	A flag indicating whether the SmartServer can be accessed via HTTP.	r/w

19.1.4 Static System Information

You can use the *SystemService_Read_Info* function to get general information about the SmartServer, including hardware and firmware versions, flash memory capacity and limits, and FPM license status. To do this, you provide the *SystemService_Read_Info* function with an <iLONSystemService> element that includes one <UCPTsystemInfoType> property that is set to SI_STATIC.

Request

```
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <UCPTsystemInfoType>SI_STATIC</UCPTsystemInfoType>
  </iLONSystemService>
</SystemService_Read_Info>
```

Response

```
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <STATIC>
        <UCPTlogTime>2008-04-03T17:29:47.314-07:00</UCPTlogTime>
        <UCPTcpuClockRate>264000000</UCPTcpuClockRate>
        <UCPTtotalDiskSpace>65398784</UCPTtotalDiskSpace>
        <UCPTmodelNumber>72102</UCPTmodelNumber>
```

```
        <UCPTmodemPresent>true</UCPTmodemPresent>
        <UCPTip852RouterPresent>true</UCPTip852RouterPresent>
        <UCPTbootromVersion>4.01.112</UCPTbootromVersion>
        <UCPTtransceiverId>TP_FT_10</UCPTtransceiverId>
        <UCPTfirmwareVersion>4.01.012</UCPTfirmwareVersion>
        <UCPTcpldVersion>74</UCPTcpldVersion>
        <UCPThwVersion>3</UCPThwVersion>
        <UCPTlonTalkUID0>3 0 0 19 7b 80</UCPTlonTalkUID0>
        <UCPTdiskSpareBlocksTotal>94</UCPTdiskSpareBlocksTotal>
        <UCPTdiskSpareBlocksMinRec>8</UCPTdiskSpareBlocksMinRec>
        <UCPTdiskEraseRateMaxRec>78</UCPTdiskEraseRateMaxRec>
        <UCPTprogrammabilityState>PS_REGISTERED</UCPTprogrammabilityState>
        <UCPTSoftwareDisabledByDownRevHardware>false</UCPTSoftwareDisabledByDownRevHardware>
    </STATIC>
  </iLONSystemService>
</SystemService_Read_Info>
```

The *SystemService_Read_Info* function returns one <STATIC> item. The following table lists and describes the properties of the <STATIC> item. Note that all of the <STATIC> item's properties are read-only.

Property	**Description**	**R/W**
`<UCPTlogTime>`	A timestamp indicating the time that the static system information was returned. This timestamp uses the ISO 8601 format, which is as follows: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm	r
`<UCPTcpuClockRate>`	The CPU speed (in MHz) of the SmartServer.	r
`<UCPTtotalDiskSpace>`	The total disk space available on the SmartServer (in bytes). The SmartServer hardware has a 64MB flash disk.	r
`<UCPTmodelNumber>`	The SmartServer's model number.	r
`<UCPTmodemPresent>`	A flag indicating whether the SmartServer has an internal modem.	r
`<UCPTip852RouterPresent>`	A flag indicating whether IP-852 routing is licensed on the SmartServer.	r
`<UCPTbootromVersion>`	The bootrom version of the SmartServer.	r
`<UCPTtransceiverID>`	The channel type of the SmartServer. For free topology models of the SmartServer, this is TP_FT_10. For power line models, this is PL_20N (or PL-20C if CENELEC is enabled).	r
`<UCPTfirmwareVersion>`	The firmware version of the SmartServer.	r
`<UCPTcpldVersion>`	The CPLD version.	r
`<UCPTlonTalkUID0>`	The first of the SmartServer's 16 Neuron IDs.	r
`<UCPTdiskSpareBlocksTotal>`	The total number of spare flash blocks initially available on the SmartServer. The SmartServer hardware contains up to 94 spare flash blocks that are	r

Property	Description	R/W
	used to accommodate block failures.	
<UCPTdiskSpareBlocksMinRec>	The recommended minimum number of spare flash blocks (8 blocks). If a flash block on the SmartServer fails and there are no spare blocks remaining, the flash disk may become unreliable. The SmartServer should be replaced before this happens.	r
<UCPTdiskEraseRateMaxRec>	The maximum rate of erases/minute of the SmartServer flash disk.	r
<UCPTprogrammabilityState>	This property specifies whether a SmartServer Programming Activation Key is installed on the SmartServer. This property may be one of the following values: • PS_REGISTERED • PS_ UNREGISTERED A SmartServer Programming Activation Key must be installed on a SmartServer in order to deploy freely programmable modules (FPMs) on it. For more information on creating and deploying FPMs, see the *i.LON SmartServer Programming Tools User's Guide*.	r

19.1.5 Real-Time System Information

You can use the *SystemService_Read_Info* function to get general network performance statistics for the SmartServer, including the current flash memory usage, CPU utilization, data point message failures, and the time of last Web browser refresh. To do this, you provide the *SystemService_Read_Info* function with an <iLONSystemService> element that includes one <UCPTsystemInfoType> property that is set to SI_REAL_TIME.

Request

```
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <UCPTsystemInfoType>SI_REAL_TIME</UCPTsystemInfoType>
  </iLONSystemService>
</SystemService_Read_Info>
```

Response

```
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <REAL_TIME>
        <UCPTlogTime>2008-04-03T17:35:45.971-07:00</UCPTlogTime>
        <UCPTminutesSinceReboot>28859</UCPTminutesSinceReboot>
        <UCPTcpuUsage>4.9342679879218938</UCPTcpuUsage>
        <UCPTfreeDiskSpace>21635072</UCPTfreeDiskSpace>
        <UCPTdiskSpareBlocks>94</UCPTdiskSpareBlocks>
        <UCPTdiskEraseRates>0 1 0 1</UCPTdiskEraseRates>
        <UCPTfreeMemory>21914320</UCPTfreeMemory>
        <UCPTmaxBlockSizeFree>13623392</UCPTmaxBlockSizeFree>
        <UCPTnumBlocksFree>724</UCPTnumBlocksFree>
        <UCPTnumBlocksAlloc>221653</UCPTnumBlocksAlloc>
        <AoUCPTdpMessageFailures>
            <UCPTdpMessageFailures>2 0</UCPTdpMessageFailures>
            <UCPTdpMessageFailures>10 0</UCPTdpMessageFailures>
            <UCPTdpMessageFailures>60 0</UCPTdpMessageFailures>
```

```
          <UCPTdpMessageFailures>1440 0</UCPTdpMessageFailures>
        </AoUCPTdpMessageFailures>
    </REAL_TIME>
  </iLONSystemService>
</SystemService_Read_Info>
```

The *SystemService_Read_Info* function returns one <REAL_TIME> item. The following table lists and describes the properties of the < REAL_TIME> item. Note that all of the <REAL_TIME> item's properties are read-only.

Property	**Description**	**R/W**
`<UCPTlogTime>`	A timestamp indicating the time that the real-time system information was returned. This timestamp uses the ISO 8601 format, which is as follows: YYYY-MM-DD**T**HH:MM:SS.sss**Z**Phh:mm	r
`<UCPTminutesSinceReboot>`	Total number of minutes that have elapsed since the SmartServer was last rebooted.	r
`<UCPTcpuUsage>`	The percentage of time the SmartServer's processor is working. This property is a primary indicator of processor activity. If your SmartServer seems to be running slowly, this property may display a higher percentage.	r
`<UCPTfreeDiskSpace>`	The current available space on the SmartServer flash disk (bytes). The SmartServer hardware has a 64MB flash disk, and initially has approximately 28 MB of free disk space.	r
`<UCPTdiskSpareBlocks>`	The total number of spare flash blocks remaining on the SmartServer. Initially, the SmartServer hardware contains up to 94 spare flash blocks that are used to accommodate block failures. However, it is normal for a small number of flash blocks to initially be marked as failed by the flash manufacturer, and additional blocks may fail after extended use, so the number of available spare blocks on your SmartServer may vary. This does not adversely affect the normal operation of the flash disk, as long as some spare blocks are available.	r
`<UCPTdiskEraseRates>`	A space-separated array listing the average rate of flash disk erases/minute over the last 3 minutes, over the last 1 hour, from the moving average calculation, and since the last reboot.	r
`<UCPTfreeMemory>`	The current available RAM (in bytes) on the SmartServer. The SmartServer initially has approximately 35 MB of free RAM.	r
`<UCPTmaxBlockSizeFree>`	The size (in bytes) of the largest block of RAM on the SmartServer.	r

Property	Description	R/W
<UCPTnumBlocksFree>	The number of free blocks of RAM on the SmartServer.	r
<UCPTnumBlocksAlloc>	The number of blocks of RAM that have been allocated on the SmartServer.	r
<AoUCPTdpMessageFailures>	An array of <UCPTdpMessageFailures> that indicate the rate and number of data point message failures over the last 2 minutes, 10 minutes, 60 minutes, and 24 hours (1440 minutes). Data point message failures occur when the SmartServer's data server passes data point updates to an application. This problem can be caused by high-network traffic, misconfigured applications, or overactive remote applications attempting read or write to the SmartServer's data points.	r

19.1.6 E-Mail Settings

You can use the *SystemService_Read_Info* function to get the settings of the e-mail (SMTP) servers connected to your SmartServer. To do this, you provide the *SystemService_Read_Info* function with an <iLONSystemService> element that includes one <UCPTsystemInfoType> property that is set to SI_MAIL.

Request

```
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <UCPTsystemInfoType>SI_MAIL</UCPTsystemInfoType>
  </iLONSystemService>
</SystemService_Read_Info>
```

Response

```
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <MAIL>
            <UCPTmailServer>10.2.120.3</UCPTmailServer>
            <UCPTmailOriginator>ilon@echelon.com</UCPTmailOriginator>
            <UCPTmailLogin>ilon</UCPTmailLogin>
            <UCPTmailPassword>ilon</UCPTmailPassword>
        </MAIL>
  </iLONSystemService>
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
```

The *SystemService_Read_Info* function returns one <MAIL> item. The following table lists and describes the properties of the <MAIL> item. You can use the *SystemService_Write_Info* function to configure those properties that are marked (r/w).

Property	Description	R/W
<UCPTmailServer>	The IP address of the designated default SMTP e-mail server.	r/w
<UCPTmailOriginator>	The string that appears in the From field of e-mail messages sent through this SMTP service (*e.g.* lonfloor1@echelon.com).	r/w

Property	Description	R/W
<UCPTmailLogin>	The user name for logging in to an SMTP server that requires authentication. The SmartServer and the SMTP server will automatically negotiate the authentication mechanism to be used (PLAIN, LOGIN, or CRAM-MD5). The SmartServer does not support the POP before SMTP authentication mechanism.	r/w
`<UCPTmailPassword>	The password used for logging in to an SMTP server that requires authentication.	r/w

19.1.7 IP-852 Router Settings

You can use the *SystemService_Read_Info* function to get the settings of the SmartServer's internal IP-852 router. To do this, you provide the *SystemService_Read_Info* function with an <iLONSystemService> element that includes one <UCPTsystemInfoType> property that is set to SI_RTR.

Request

```
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <UCPTsystemInfoType>SI_RTR</UCPTsystemInfoType>
  </iLONSystemService>
</SystemService_Read_Info>
```

Response

```
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <RTR>
        <UCPTrouterType>Configured</UCPTrouterType>
        <UCPTrouterAuthKey>00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00</UCPTrouterAuthKey>
        <UCPTip852ConfigServerAddr>10.2.124.77</UCPTip852ConfigServerAddr>
        <UCPTip852ConfigServerPort>1629</UCPTip852ConfigServerPort>
        <UCPTip852LocalPort>1628</UCPTip852LocalPort>
        <UCPTrouterLonWorksAddr_IP>B3/2/1</UCPTrouterLonWorksAddr_IP>
        <UCPTrouterLonWorksAddr_LT>B3/1/2</UCPTrouterLonWorksAddr_LT>
    </RTR>
  </iLONSystemService>
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
```

The *SystemService_Read_Info* function returns one <RTR> item. The following table lists and describes the properties of the <RTR> item. You can use the *SystemService_Write_Info* function to configure those properties that are marked (r/w).

Property	Description	R/W
<UCPTrouterType>	• This property specifies the router type. You can use the *SystemService_Write_Info* function to change the IP-852 router to a Repeater. You cannot change the IP-852 router to any other router type. • Configured. The router determines which packets to forward based on internal routing tables. These routing tables contain one entry for each subnet in the application domain. Whenever a router receives a packet, it examines the source and destination subnet ID to determine whether to forward the packet. This is the recommended type because it optimizes network traffic and enables the channels	r

Property	Description	R/W
	on which devices are attached to be determined automatically. Configured routers also support the use of redundant routers (multiple routers connecting two channels), which provide for redundant message paths and greater system reliability. • Learning. Like a configured router, the router determines which packets to forward based on internal routing tables. Learning routers, though, have their routing tables stored in volatile memory; therefore, the router forwards packets addressed to all subnets in the application domain after being reset. Whenever a learning router receives a packet from one of its channels, it uses the source subnet ID to learn the network topology. It sets the corresponding routing table entries to indicate that the subnet in question is to be found in the direction from which the packet was received. A learning router always forwards all group-addressed messages. • Repeater. The router forwards all valid packets received on one channel to the other channel. Subnets cannot span non-permanent repeaters. You can use a non-permanent repeater to maintain flexibility in order to change the router type later. • Bridge. The router forwards all valid packets that match the network domain. Subnets cannot span non-permanent bridges. You can use a non-permanent bridge to maintain flexibility in order to change the router type later. • Unknown. The SmartServer automatically select the appropriate router type.	
<UCPTrouterAuthKey>	A 16-digit hexadecimal MD5 authentication key to be used for the IP-852 channel.	r/w
<UCPTip852ConfigServerAddr>	The IP address of the IP-852 Configuration Server connected to the SmartServer.	r/w
<UCPTip852ConfigServerPort>	The port used by the IP-852 Configuration Server to receive messages from the SmartServer. The default port is 1629.	r/w
<UCPTip852LocalPort>	The port used by the SmartServer to receive messages from the IP-852 Configuration Server. The default port is 1628.	r/w
<UCPTrouterLonWorksAddr_IP>	The domain, subnet, node address of the IP-852 router on the IP channel connected to the SmartServer.	r

Property	Description	R/W
<UCPTrouterLonWorksAddr_LT>	The domain, subnet, node address of the IP-852 router on the LonTalk channel connected to the SmartServer.	r

19.1.8 IP-852 Router Statistics

You can use the *SystemService_Read_Info* function to get network performance statistics for the SmartServer's internal IP-852 router. To do this, you provide the *SystemService_Read_Info* function with an <iLONSystemService> element that includes one <UCPTsystemInfoType> property that is set to SI_RTR_STAT

Request

```
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <UCPTsystemInfoType>SI_RTR_STAT</UCPTsystemInfoType>
  </iLONSystemService>
</SystemService_Read_Info>
```

Response

```
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <RTR_STAT>
        <UCPTlastClear>2008-04-04T17:14:44.840-07:00</UCPTlastClear>
        <UCPTltPacketsSent>11</UCPTltPacketsSent>
        <UCPTltPacketsReceived>2444</UCPTltPacketsReceived>
        <UCPTltPacketsLost>0</UCPTltPacketsLost>
        <UCPTipPacketsSent>0</UCPTipPacketsSent>
        <UCPTipPacketsReceived>0</UCPTipPacketsReceived>
        <UCPTipPacketsStale>0</UCPTipPacketsStale>
        <UCPTcfgPacketsSent>20</UCPTcfgPacketsSent>
        <UCPTcfgPacketsReceived>20</UCPTcfgPacketsReceived>
    </RTR_STAT>
  </iLONSystemService>
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
```

The *SystemService_Read_Info* function returns one <RTR_STAT> item. The following table lists and describes the properties of the <RTR_STAT> item. Note that all of the <RTR_STAT> item's properties are read-only.

Property	Description	R/W
<UCPTltPacketsSent>	The number of LonTalk packets transmitted by the SmartServer's IP-852 router.	r
<UCPTltPacketsReceived>	The number of LonTalk packets received by the SmartServer's IP-852 router.	r
<UCPTltPacketsLost	The number of LonTalk packets lost.	r
<UCPTipPacketsSent>	The number of IP packets transmitted by the SmartServer's IP-852 router.	r
<UCPTipPacketsReceived>	The number of IP packets received by the SmartServer's IP-852 router.	r
<UCPTipPacketsStale>	The number of IP packets lost.	r
<UCPTcfgPacketsSent>	Packets sent by the SmartServer's IP-852 router to the	r

Property	Description	R/W
	IP-852 Configuration Server on the IP-852 channel.	
<UCPTcfgPacketsReceived>	Packets received by the SmartServer's IP-852 router from the IP-852 Configuration Server on the IP-852 channel.	R

19.1.9 LonScanner Protocol Analyzer

You can use the *SystemService_Read_Info* function to get the SmartServer's LonScanner Protocol Analyzer (LSPA) settings. To do this, you provide the *SystemService_Read_Info* function with an <iLONSystemService> element that includes one <UCPTsystemInfoType> property that is set to SI_LSPA.

Request

```
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <UCPTsystemInfoType>SI_LSPA</UCPTsystemInfoType>
  </iLONSystemService>
</SystemService_Read_Info>
```

Response

```
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <LSPA>
        <UCPTlspaPort>1629</UCPTlspaPort>
        <UCPTlspaEnable>true</UCPTlspaEnable>
        <UCPTlspaCaptureAllPackets>false</UCPTlspaCaptureAllPackets>
    </LSPA>
  </iLONSystemService>
<SystemService_Read_Info xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
```

The *SystemService_Read_Info* function returns one < SI_LSPA> item. The following table lists and describes the properties of the < SI_LSPA> item. You can use the *SystemService_Write_Info* function to configure those properties that are marked (r/w).

Property	Description	R/W
<UCPTlspaPort>	The port the SmartServer uses for communication with the LonScanner Protocol Analyzer tool.	r/w
<UCPTlspaEnable>	This property enables the SmartServer to be connected to the LonScanner Protocol Analyzer tool, which you can use to monitor and diagnose network traffic. To enable the SmartServer to be connected to the LonScanner Protocol Analyzer tool, set this property to "true". To disable this feature, set this property to "false".	r/w

Property	Description	R/W
<UCPTlspaCaptureAllPackets>	This property enables packets directly transmitted to the internal devices on the SmartServer to be viewed with the LonScanner Protocol Analyzer tool. These packets will still not be sent on the physical network. To enable packets sent by the internal devices on the SmartServer to be viewed with the LonScanner Protocol Analyzer tool, set this property to "true". To disable this feature, set this property to "false".	r/w

19.1.10 Reboot

The system module includes a *SystemService_Reboot* function that you can use to reboot a SmartServer.

Request

```
<SystemService_Reboot xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/" />
```

Response

```
<SystemService_RebootResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
    <Result />
</SystemService_RebootResponse>
```

19.2 System Test Methods

You can use the *SystemService_Test* function to test SMTP mail servers, test the connections between the SmartServer and IP-852 Configuration servers, and to test connections between the SmartServer and host devices such as SMTP mail servers, SNTP time servers, remote SmartServers, and LNS Servers.

The *SystemService_Test* function takes a string consisting of an <iLONSystemService> element that includes one <Test> element. The <Test> element has a <UCPTtestType> property that specifies an enumeration representing the type of test to be performed (SMTP CONNECTION_TEST, CONFIG_SERVER_TEST, or CONNECTION_TEST) and other properties (required or optional) specific to the type of test being performed.

The *SystemService_Test* function returns a string containing <iLONSystemService> element that includes a single <RESULTS> item listing the results of the test.

The following example demonstrates how to use the *SystemService_Test* function in your code:

```
string testData = "<iLONSystemService><Test><UCPTTestType>SMTP_TEST<UCPTTestType>
<UCPTstate>ST_BEGIN</UCPTstate></Test></iLONSystemService>"
```

19.2.1 SMTP E-Mail Server Test

You can use the *SystemService_Test* function to test that the default SMTP e-mail server that is connected to the SmartServer can send out e-mail messages. To do this, you provide the *SystemService_Test* function with an <iLONSystemService> element that includes one <Test> element that specifies the following properties:

Property	Description	Required/ Optional
<UCPTTestType>	Enumeration that defines the type of test to perform. This property must be set to SMTP_TEST.	Required

<UCPTstate>	This property specifies whether to start a test or get the current status of the test. This property may be set to one of the following values: • ST_BEGIN. Starts the test. • ST_STATUS. Returns the status of the test.	Required
<UCPTemailAddress>	A string specifying the e-mail address to where the test e-mail will be sent.	Optional
<UCPToriginator>	A string specifying the originator of the e-mail message.	Optional
<UCPTemailSubject>	A string specifying the subject line of the test e-mail message.	Optional
<UCPTemailFormat>	A string specifying the text of the test e-mail message. Note that you may not include flags such as those used by the Alarm Notifier application.	Optional
<UCPTemailAttachment>	A string specifying a full path on the SmartServer flash disk of a file to be attached to the test e-mail message such as "/root/AlarmLog/sumlog1.csv" if you were attaching an alarm log.	Optional
UCPTcount	This property specifies the number of times the e-mail message will be sent.	Optional

Note: If you are checking the status of a test (<UCPTstate> is set to ST_STATUS), you only need to provide the <UCPTTestType> and <UCPTstate> properties.

The following examples demonstrate how to use the *SystemService_Test* function to test that the default SMTP e-mail server can send out e-mail messages and check the status of the test:

Request (test the SMTP e-mail server)

```
<SystemService_Test xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <Test>
        <UCPTtestType>SMTP_TEST</UCPTtestType>
       <UCPTstate>ST_BEGIN</UCPTstate>
       <UCPTemailAddress>ilonuser@echelon.com</UCPTemailAddress>
       <UCPToriginator>sender</UCPToriginator>
        <UCPTemailSubject>This is a test</UCPTemailSubject>
       <UCPTemailFormat></UCPTemailFormat>
        <UCPTemailAttachment></UCPTemailAttachment>
     </Test>
  </iLONSystemService>
</SystemService_Test>
```

Request (check the status of the SMTP e-mail server test)

```
<SystemService_Test xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <Test>
        <UCPTtestType>SMTP_TEST</UCPTtestType>
       <UCPTstate>ST_STATUS</UCPTstate>
    </iLONSystemService>
</SystemService_Test>
```

Response

```
<SystemService_TestResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <Result>
    <iLONSystemService>
        <Test>
            <UCPTtestType>SMTP_TEST</UCPTtestType>
            <UCPTlastUpdate>2008-04-10T15:40:37.488-07:00</UCPTlastUpdate>
            <UCPTstate>IDLE</UCPTstate>
            <UCPTerror>0</UCPTerror>
            <UCPTdetailDescr></UCPTdetailDescr>
            <AoTestLog>
                <TestLog>
                    <UCPTindex>0</UCPTindex>
                    <UCPTlogTime>2008-04-10T15:36:44.202-07:00</UCPTlogTime>
                    <UCPTdescription>SMTPsend</UCPTdescription>
                    <UCPTerror>0</UCPTerror>
                    <UCPTdetailDescr>ilonuser@echelon.com</UCPTdetailDescr>
                </TestLog>
                <TestLog>
                    <UCPTindex>1</UCPTindex>
                    <UCPTlogTime>2008-04-10T15:36:45.202-07:00</UCPTlogTime>
                    <UCPTdescription>Connection released</UCPTdescription>
                    <UCPTerror>0</UCPTerror>
                    <UCPTdetailDescr>10.2.120.87</UCPTdetailDescr>
                </TestLog>
            </AoTestLog>
        </Test>
  </iLONSystemService>
 </Result>
<SystemService_TestResponse>
```

The *SystemService_Test* function returns a <iLONSystemService> element that includes a single <RESULTS> item that has the following properties.

Note: These properties also apply to the Reponses to IP-852 Configuration Server tests, connection tests, and test status checks.

Property	**Description**
<UCPTtestType>	An enumeration that defines the type of test performed, which can be one of the following values: • SMTP CONNECTION_TEST • CONFIG_SERVER_TEST • CONNECTION_TEST
<UCPTlastUpdate>	Time stamp of the beginning of the last connection test (in ISO 8601 format with an offset). When this message is a response to a successful test, this time stamp displays the start time of the current test; therefore, this timestamp may be used to match status responses to a specific begin message.
<UCPTstate>	An enumeration specifying the status of the test. This property may be one of the following values: • SS_IDLE • SS_CONNECTING • SS_DISCONNECTING

	• SS_SENDING_MAIL • SS_BUSY
<UCPTerror>	A flag indicating whether there was an error with the test.
<UCPTdetailDescr>	A string that displays the last SMTP error received from either the server or client. This may be implementation specific between SMTP servers, so the entire string is copied to this parameter.
<AoTestLog>	An array of connection log entries <UCPTtestLog>. Each <UCPTtestLog> entry has the following properties: • <UCPTindex>. The index value assigned to the log entry. • <UCPTlogTime>. The time the entry was recorded. • <UCPTdescription>. A description of the action performed. This may have the following values: • SMTP tests: "Send SMTP" or "Connection Released". • IP-852 Configuration Server tests: "Connection established", Request configuration server status", "Configuration server is not responding", or "connection released" • Connection tests: "Create Socket", "Connect Socket ", or "Close Socket". • <UCPTerror>. A flag indicating whether there was an error with the test. • <UCPTdetailDescr>. A string that displays the IP address of the host device being tested or the value specified in the <UCPTemailAddress> property of an SMTP test.

19.2.2 IP-852 Configuration Server Test

You can use the *SystemService_Test* function to test the connections between the SmartServer and the default IP-852 Configuration Server. To do this, you provide the *SystemService_Test* function with an <iLONSystemService> element that includes one <Test> element. The <Test> element must specify a <UCPTtestType> that is set to CONFIG_SERVER_TEST, and it must specify a <UCPTstate> property that is set to ST_BEGIN (to start the test) or ST_STATUS (to check the status of the test).

Request (test the connection to an IP-852 Configuration Server)

```
<SystemService_Test xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <Test>
        <UCPTtestType>CONFIG_SERVER_TEST</UCPTtestType>
       <UCPTstate>ST_STATUS</UCPTstate>
    </Test>
  </iLONSystemService>
```

```
</SystemService_Test>
```

Request (check the status of the test)

```
<SystemService_Test xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <Test>
        <UCPTtestType>CONFIG_SERVER_TEST</UCPTtestType>
       <UCPTstate>ST_STATUS</UCPTstate>
    </Test>
  </iLONSystemService>
</SystemService_Test>
```

Response

```
<SystemService_TestResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
 <Result>
  <iLONSystemService>
    <Test>
      <UCPTtestType>CONFIG_SERVER_TEST</UCPTtestType>
      <UCPTlastUpdate>2008-04-10T15:51:04.350-07:00</UCPTlastUpdate>
      <UCPTstate>IDLE</UCPTstate>
      <UCPTerror>0</UCPTerror>
      <UCPTdetailDescr></UCPTdetailDescr>
      <AoTestLog>
        <TestLog>
          <UCPTindex>0</UCPTindex>
          <UCPTlogTime>2008-04-10T15:50:48.350-07:00</UCPTlogTime>
          <UCPTdescription>Connection established</UCPTdescription>
          <UCPTerror>0</UCPTerror>
          <UCPTdetailDescr></UCPTdetailDescr>
        </TestLog>
        <TestLog>
          <UCPTindex>1</UCPTindex>
          <UCPTlogTime>2008-04-10T15:50:48.350-07:00</UCPTlogTime>
          <UCPTdescription>Request configuration server status</UCPTdescription>
          <UCPTerror>0</UCPTerror>
          <UCPTdetailDescr>10.2.124.77</UCPTdetailDescr>
        </TestLog>
        <TestLog>
          <UCPTindex>2</UCPTindex>
          <UCPTlogTime>2008-04-10T15:50:54.350-07:00</UCPTlogTime>
          <UCPTdescription>Configuration server is not responding</UCPTdescription>
          <UCPTerror>39</UCPTerror>
          <UCPTdetailDescr>10.2.124.77</UCPTdetailDescr>
        </TestLog>
        <TestLog>
          <UCPTindex>3</UCPTindex>
          <UCPTlogTime>2008-04-10T15:50:54.360-07:00</UCPTlogTime>
          <UCPTdescription>Connection released</UCPTdescription>
          <UCPTerror>0</UCPTerror>
          <UCPTdetailDescr>10.2.124.77</UCPTdetailDescr>
        </TestLog>
      </AoTestLog>
    </Test>
   </iLONSystemService>
  </Result>
<SystemService_TestResponse>
```

19.2.3 Connection Test

You can use the *SystemService_Test* function to test the connections between the SmartServer and host devices such as SMTP mail servers, SNTP time servers, remote SmartServers, LNS Servers, and WebBinder Target Servers. To do this, you provide the *SystemService_Test* function with an <iLONSystemService> element that includes one <Test> element that has the following properties:

Property	Description	Required/ Optional
<UCPTTestType>	Enumeration that defines the type of test to perform. This property must be set to CONNECTION_TEST.	Required
<UCPTstate>	This property specifies whether to start a test or get the current status of the test. This property may be set to one of the following values: • ST_BEGIN. Starts the test. • ST_STATUS. Returns the status of the test.	Required
<UCPThostURL>	The IP address or hostname of the host device whose connection to the SmartServer is to be tested.	Required
<UCPTport>	The port the SmartServer uses to communicate with the host device whose connection to the SmartServer is to be tested.	Optional

Note: If you are checking the status of a test (<UCPTstate> is set to ST_STATUS), you only need to provide the <UCPTTestType> and <UCPTstate> properties.

The following examples demonstrate how to use the *SystemService_Test* function to test a connection with another host device (a remote SmartServer) and check the status of the test.

Request (test the connection to a Remote SmartServer)

```
<SystemService_Test xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
   <Test>
       <UCPTtestType>CONNECTION_TEST</UCPTtestType>
       <UCPTstate>ST_BEGIN</UCPTstate>
       <UCPThostURL>10.2.124.53</UCPThostURL>
       <UCPTport>80</UCPTport>
    </Test>
  </iLONSystemService>
</SystemService_Test>
```

Request (check the status of the test)

```
<SystemService_Test xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
  <iLONSystemService>
    <Test>
        <UCPTtestType> CONNECTION_TEST</UCPTtestType>
       <UCPTstate>ST_STATUS</UCPTstate>
    </Test>
  </iLONSystemService>
</SystemService_Test>
```

Response

```
<SystemService_TestResponse xmlns="http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/">
 <Result>
  <iLONSystemService>
    <Test>
      <UCPTtestType>CONNECTION_TEST</UCPTtestType>
      <UCPTlastUpdate>2008-04-10T16:18:29.540-07:00</UCPTlastUpdate>
      <UCPTstate>IDLE</UCPTstate>
      <UCPTerror>0</UCPTerror>
```

```
      <UCPTdetailDescr></UCPTdetailDescr>
      <AoTestLog>
        <TestLog>
          <UCPTindex>0</UCPTindex>
          <UCPTlogTime>2008-04-10T16:18:25.550-07:00</UCPTlogTime>
          <UCPTdescription>Create socket</UCPTdescription>
          <UCPTerror>0</UCPTerror>
          <UCPTdetailDescr>10.2.124.82:80</UCPTdetailDescr>
        </TestLog>
        <TestLog>
          <UCPTindex>1</UCPTindex>
          <UCPTlogTime>2008-04-10T16:18:25.560-07:00</UCPTlogTime>
          <UCPTdescription>Connect socket</UCPTdescription>
          <UCPTerror>0</UCPTerror>
          <UCPTdetailDescr>10.2.124.82:80</UCPTdetailDescr>
        </TestLog>
        <TestLog>
          <UCPTindex>2</UCPTindex>
          <UCPTlogTime>2008-04-10T16:18:25.560-07:00</UCPTlogTime>
          <UCPTdescription>Close socket</UCPTdescription>
          <UCPTerror>0</UCPTerror>
          <UCPTdetailDescr>10.2.124.82:80</UCPTdetailDescr>
        </TestLog>
        <TestLog>
          <UCPTindex>3</UCPTindex>
          <UCPTlogTime>2008-04-10T16:18:25.560-07:00</UCPTlogTime>
          <UCPTdescription>Connection released</UCPTdescription>
          <UCPTerror>0</UCPTerror>
          <UCPTdetailDescr>10.2.124.82</UCPTdetailDescr>
        </TestLog>
      </AoTestLog>
    </Test>
   </iLONSystemService>
  </Result>
<SystemService_TestResponse>
```

20 Using the SOAP Interface as a Web Service

This chapter assumes that you have some familiarity with Web services programming, and that you are using the Microsoft Visual Studio .NET development environment. All sample code in this chapter is written in Visual Basic .NET and Visual C# .NET using Microsoft Visual Studio 2008. However, you can use any development tool that is able to call standard Web services with the SmartServer SOAP/XML interface.

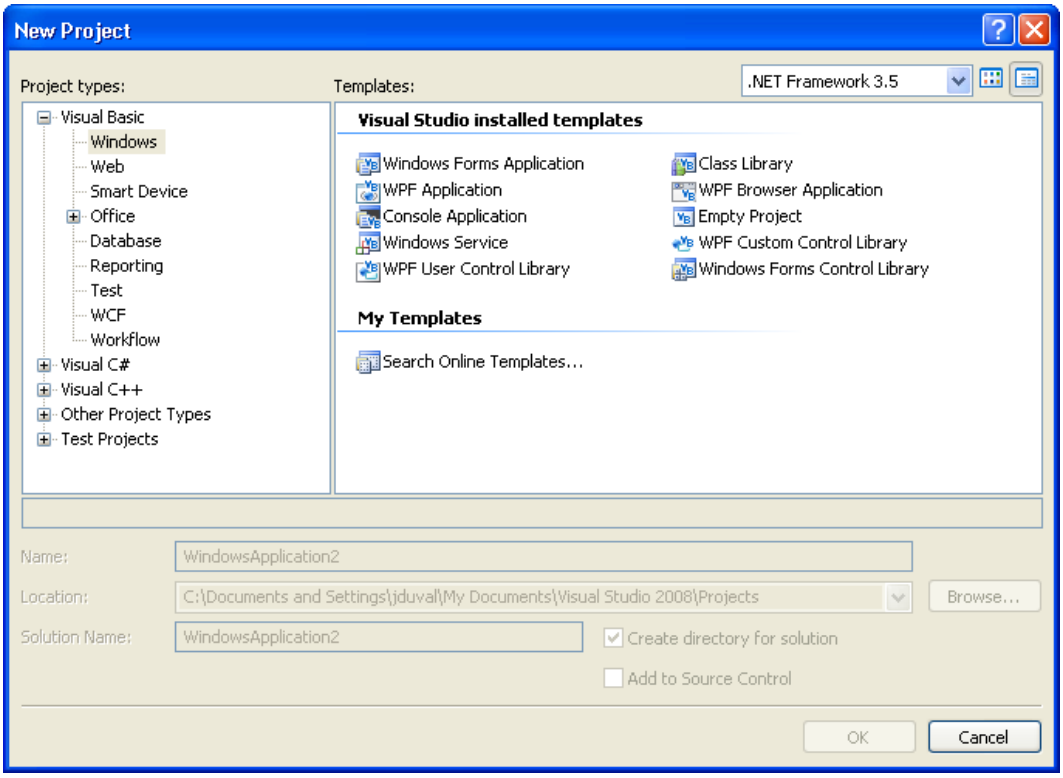
20.1 Referencing and Inheriting from the WSDL

You can use the SOAP interface as a reference with Microsoft Visual Studio .NET Framework 3.5 or .NET Framework 2.0, and create an application to modify the configuration of your SmartServer. Some development tools can import the *i.LON SmartServer WSDL File* and automatically build a class structure for sending and receiving each message. The following sections describe how to use Visual Studio .NET Framework 3.5 and .NET Framework 2.0 to reference the SmartServer SOAP interface and then inherit from the reference.

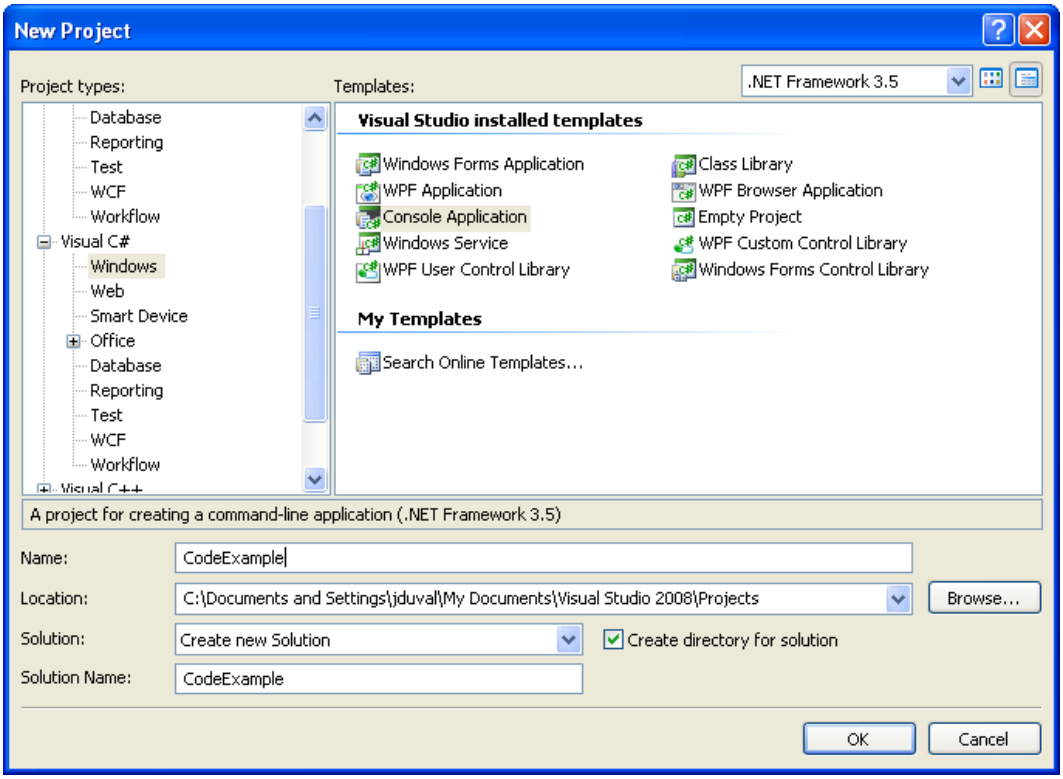
20.1.1 Referencing and Inheriting from the WSDL Using .NET 3.5 Framework

The following procedure describes how to use Visual Studio 2008 .NET Framework 3.5 to reference the SOAP interface and then inherit from the service reference.

1. Open the Microsoft Visual Studio .NET development environment.
2. Click **File**, point to **New**, and then click **Project**. The **New Project** dialog opens.

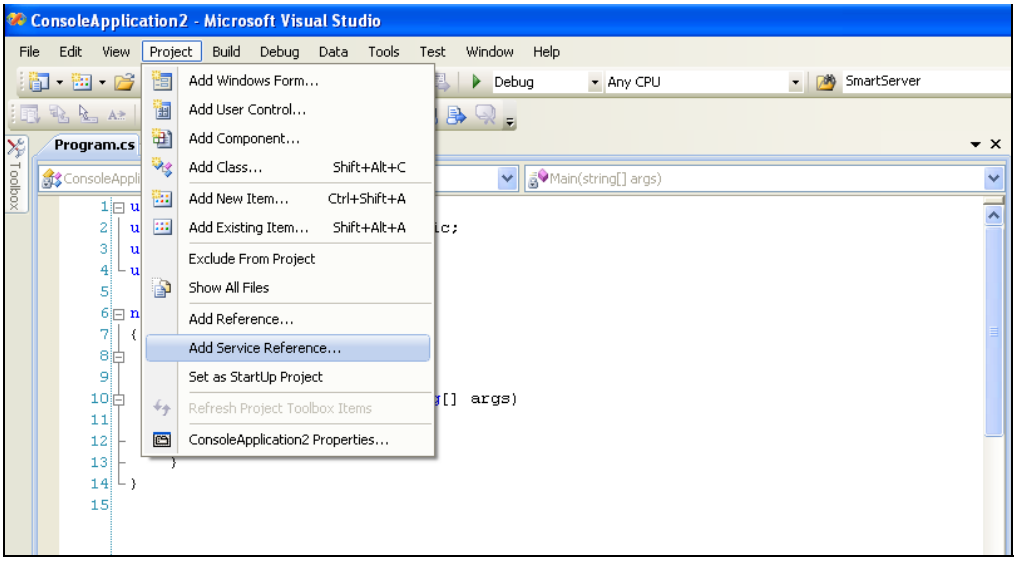


3. Enter a name, location, and project type for the project, and then click **OK**.

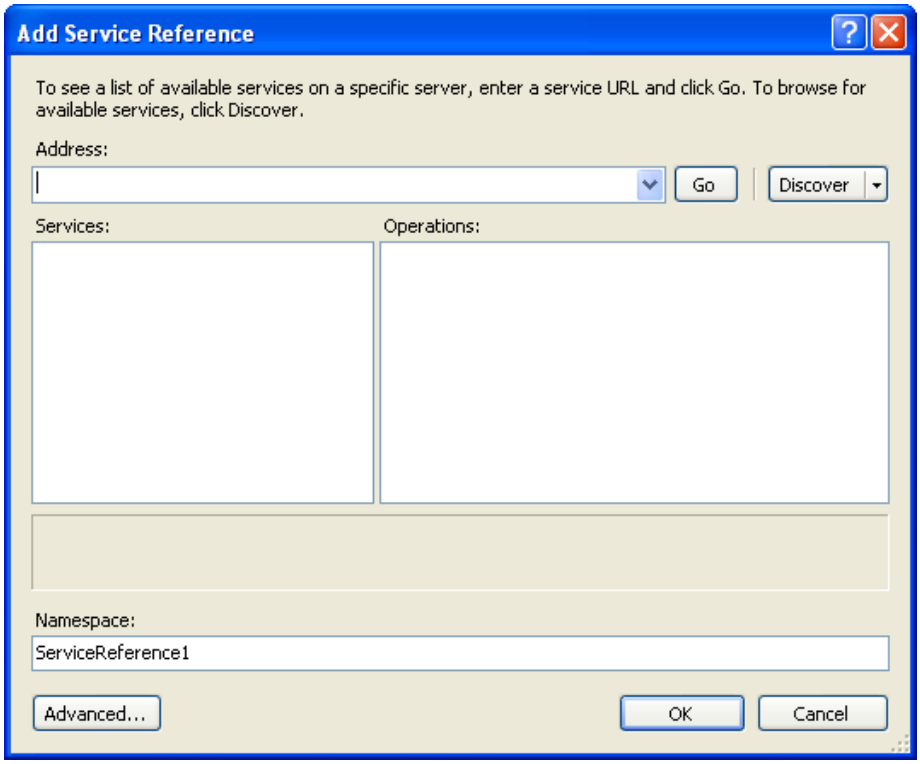


4. Add a service reference to the version 4.0 WSDL to your project. To do this, follow these steps:

   a. Click **Project** and then click **Add Service Reference**.



   b. The **Add Service Reference** dialog opens.



c. In the **URL** or **Address** box, enter the following address:

http://*SmartServer IP address***/WSDL/v4.0/iLON100.WSDL**

SmartServer IP address represents the IP address of the SmartServer your application is to reference.

If you are using the system information methods to read system information and modify the SmartServer's configuration, enter the following address in the **URL** or **Address** box:

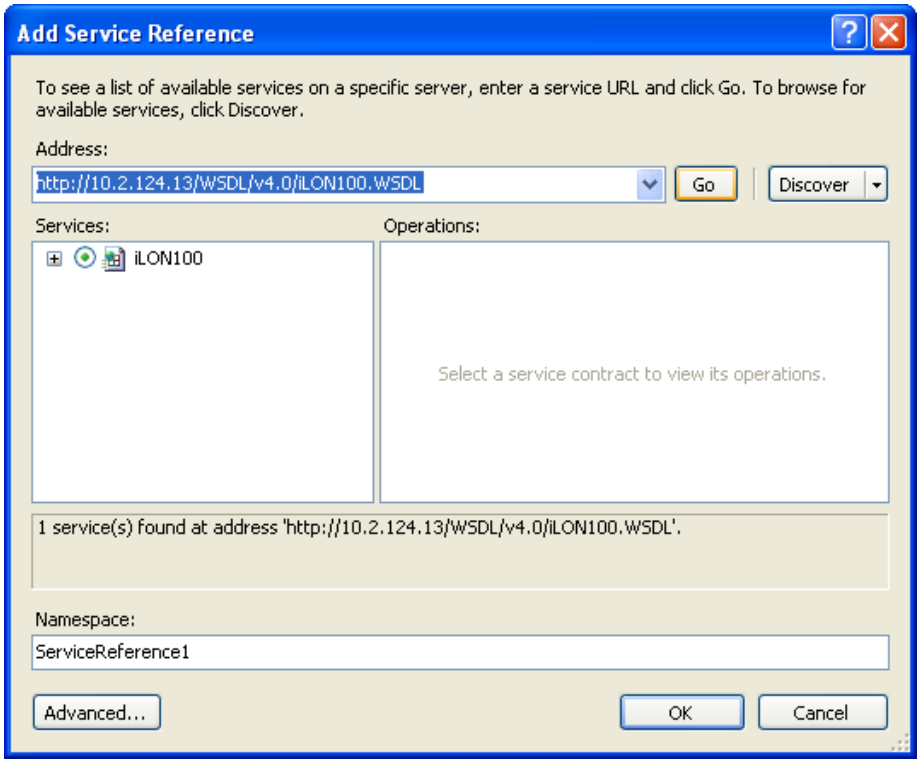
http://*SmartServer IP address***/WSDL/v4.0/iLON100_System.wsdl**

See Chapter 19 for more information on using the system information methods, and see Chapter 22 for sample code demonstrating how to use the system information methods.

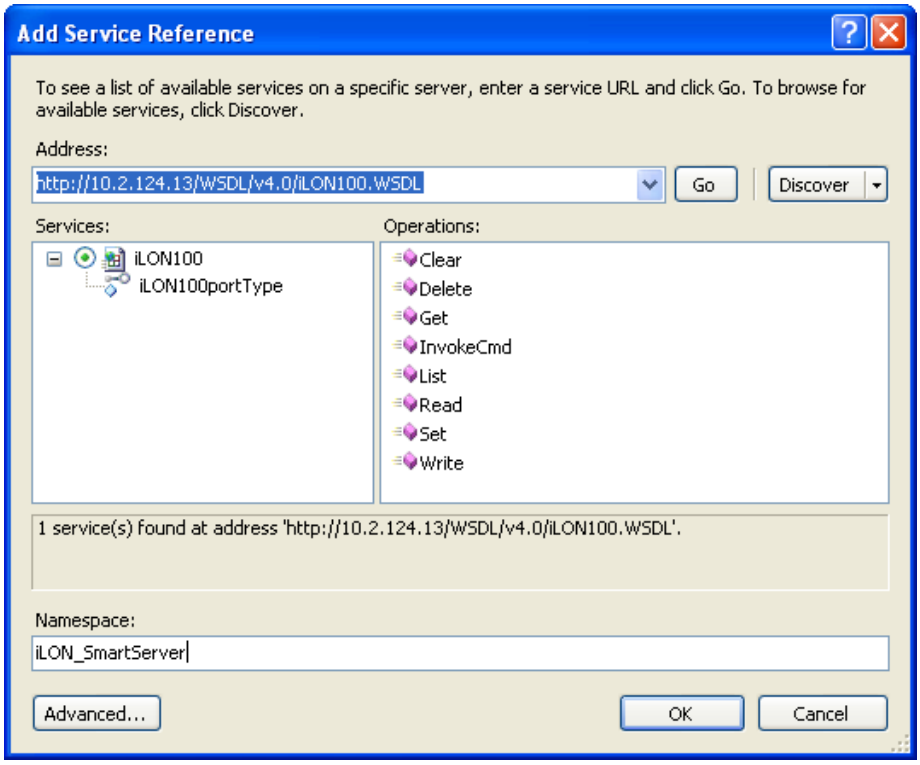
Note: All examples in this section can also be applied to LNS Proxy Web service. The LNS Proxy supports the same WSDL as the SmartServer Web service; therefore, the same programming model can be applied to program a LNS network database in a transparent manner. To program a LNS Proxy Web service instead of a SmartServer, enter the URL of the LNS Proxy in this form:

http://*LNS Proxy IP address***/LnsProxy/LnsProxyService?wsdl**

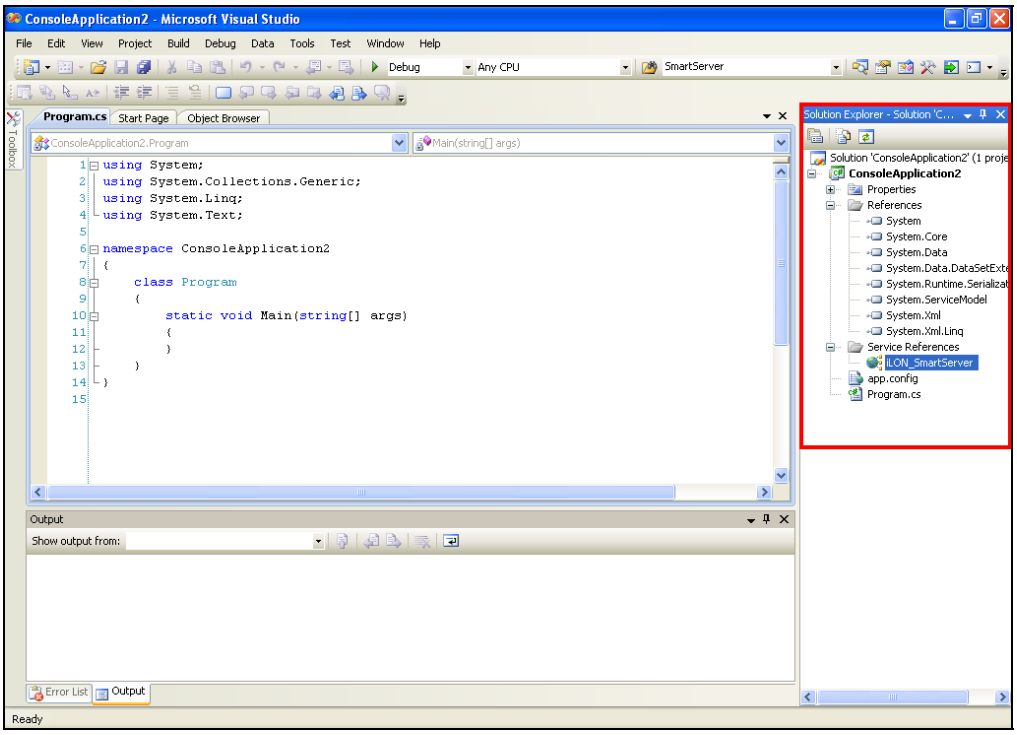
d. Click **Go**. The SmartServer's iLON100 WSDL file is displayed.



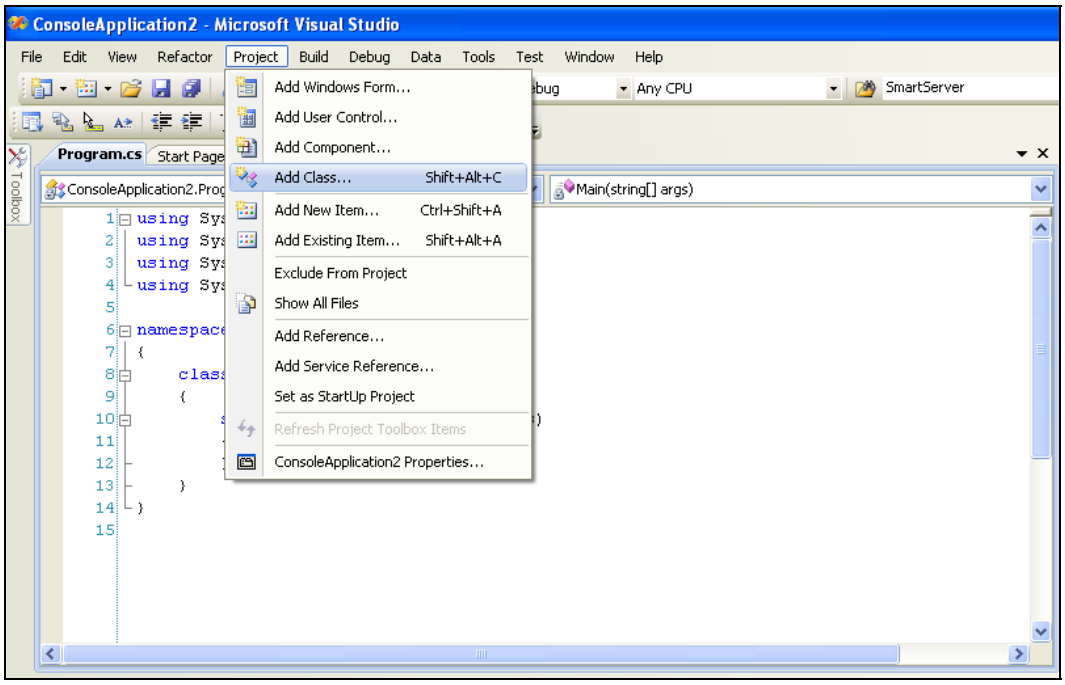
e. In the **Namespace** box, enter a name for the service reference. You will use this name when you instantiate the Web services object because it becomes a name for the proxy class that is generated automatically by Visual Studio .NET. This is described in more detail in the next section. The name used for the service reference in this example is "iLON_SmartServer".



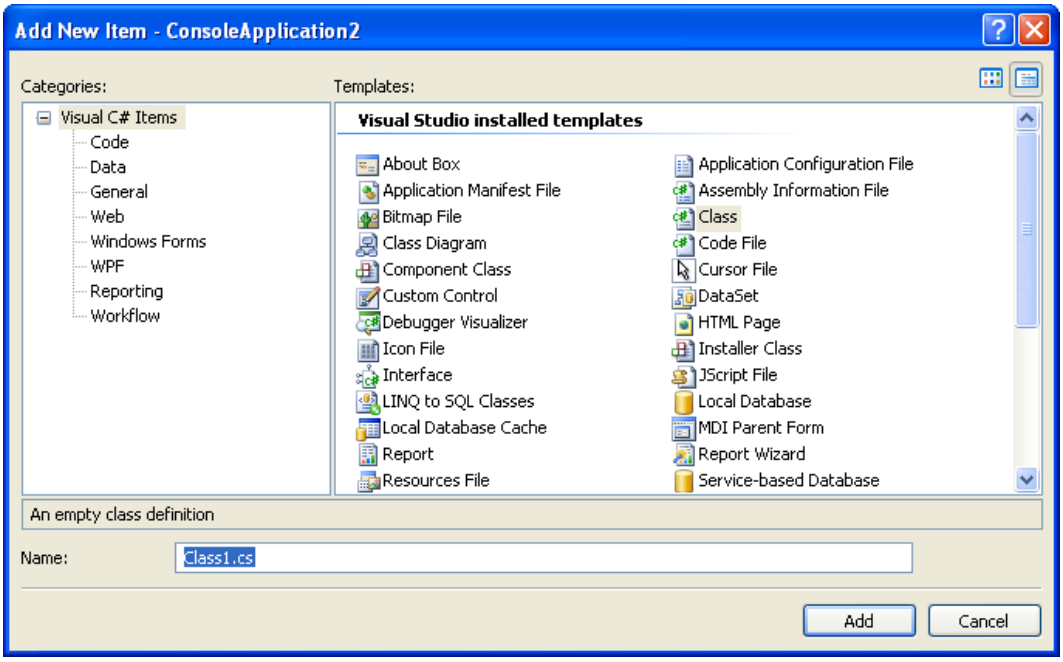
f. Click **OK**. The new service reference appears in the list of references in the **Solution Explorer** pane.



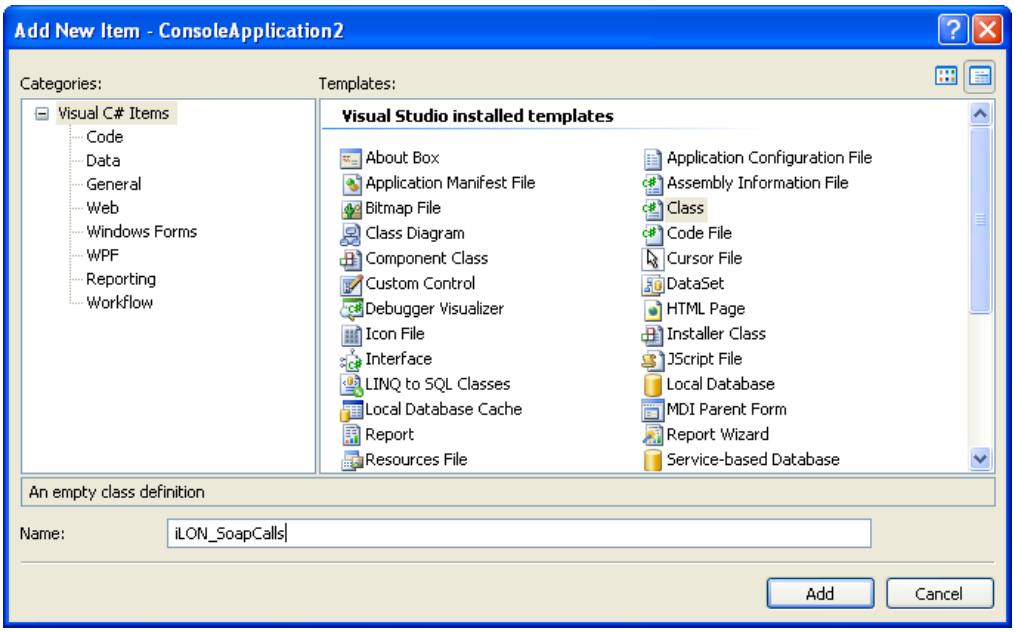
5. Click **Project**, and then click **Add Class**.



6. The **Add New Item** dialog opens.



7. In the **Name** box, enter "iLON_SoapCalls" and then click **Add.**

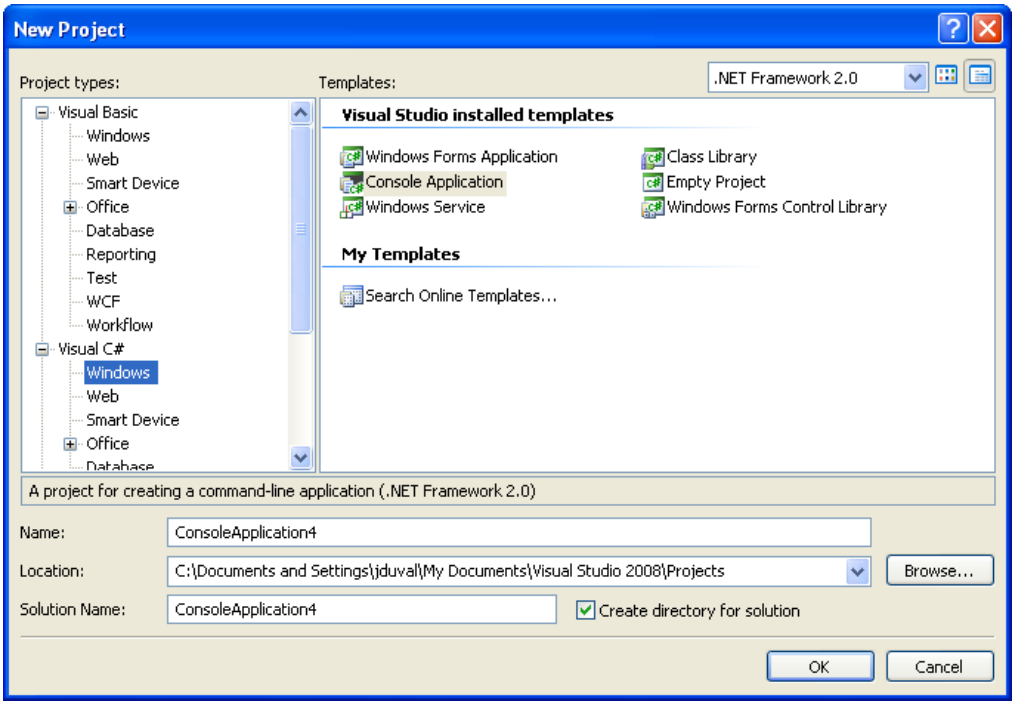


8. You should now use the **iLON_SoapCalls** class to instantiate the SmartServer Web service as described in the section 20.2.1, *Instantiating the Web Service Client in Visual C# .NET 3.5* or section 20.2.3, *Instantiating the Web Service Client in Visual Basic .NET 3.5*.

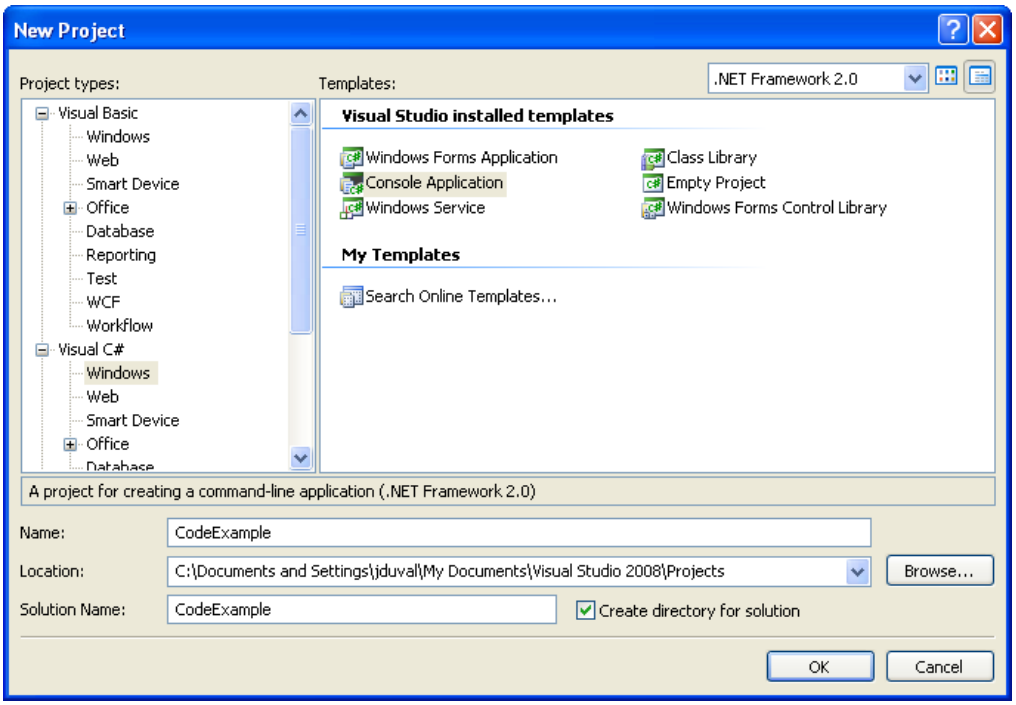
20.1.2 Referencing and Inheriting from the WSDL Using .NET 2.0 Framework

The following procedure describes how to use Visual Studio 2008 .NET Framework 2.0 to reference the SOAP interface and then inherit from the service reference.

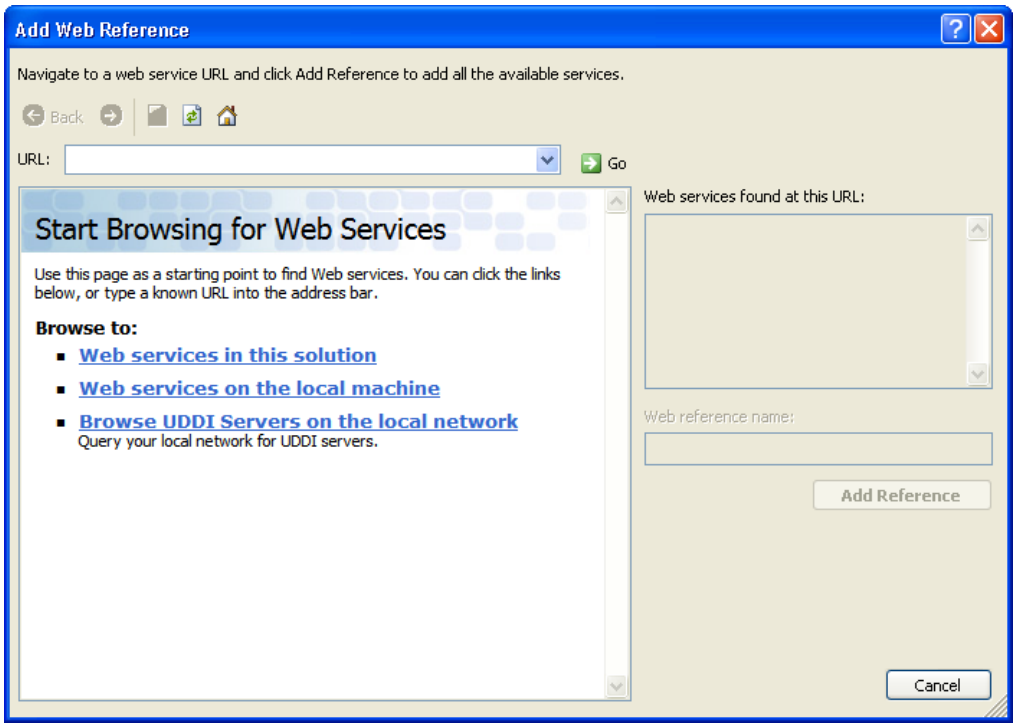
1. Open the Microsoft Visual Studio .NET development environment.
2. Click **File**, point to **New**, and then click **Project**. The **New Project** dialog opens.



3. Enter a name, location, and project type for the project, and then click **OK**.



4. Add a Web reference to the version 4.0 WSDL to your project. To do this, follow these steps:
   a. Click **Project** and then click **Add Web Reference**. The **Add Web Reference** dialog opens.



b. In the **URL** or **Address** box, enter the following address:

http://*SmartServer IP address***/WSDL/v4.0/iLON100.WSDL**

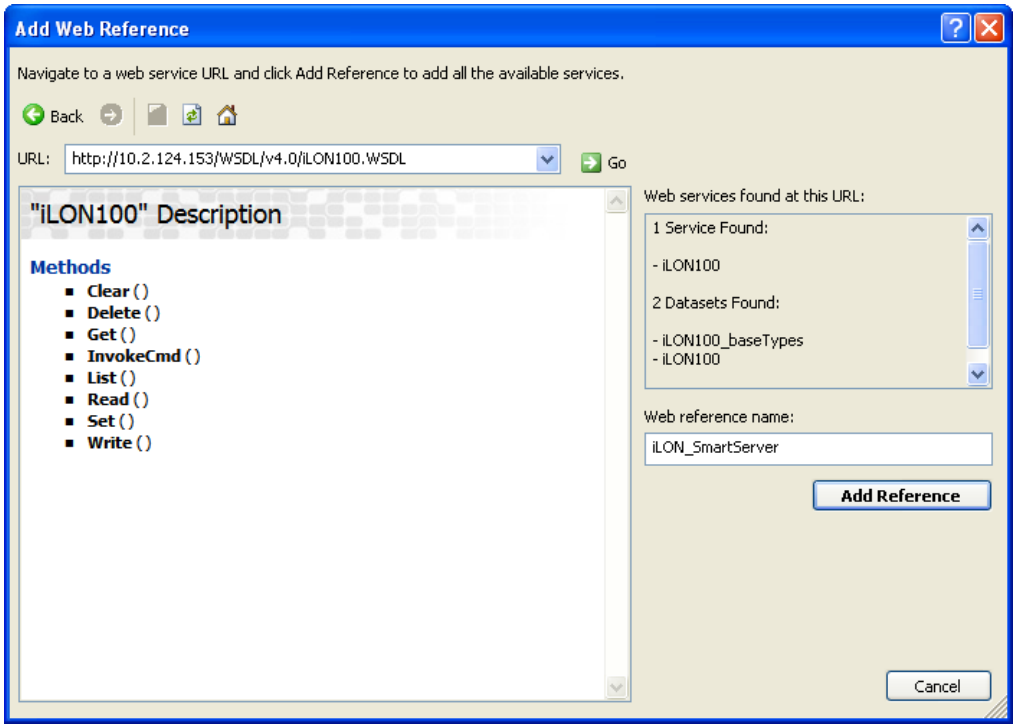
SmartServer IP address represents the IP address of the SmartServer your application is to reference.

Note: All examples in this section can also be applied to LNS Proxy Web service. The LNS Proxy supports the same WSDL as the SmartServer Web service; therefore, the same programming model can be applied to program a LNS network database in a transparent manner. To program a LNS Proxy Web service instead of a SmartServer, enter the URL of the LNS Proxy in this form:

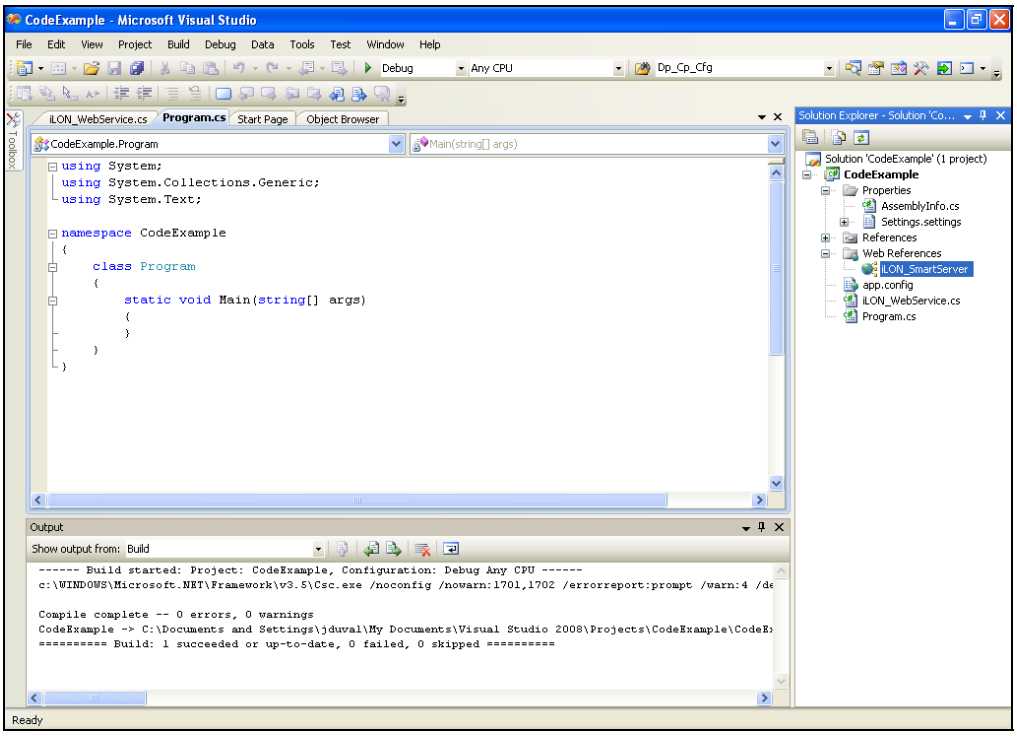
http://*LNS Proxy IP address***/LnsProxy/LnsProxyService?wsdl**

c. Click **Go**.

d. In the **Web Reference Name** box, enter a name for the Web reference. You will use this name when you instantiate the Web services object because it becomes a name for the proxy class that is generated automatically by Visual Studio .NET. This is described in more detail in the next section. This example uses “iLON_SmartServer” for the name of the Web reference.

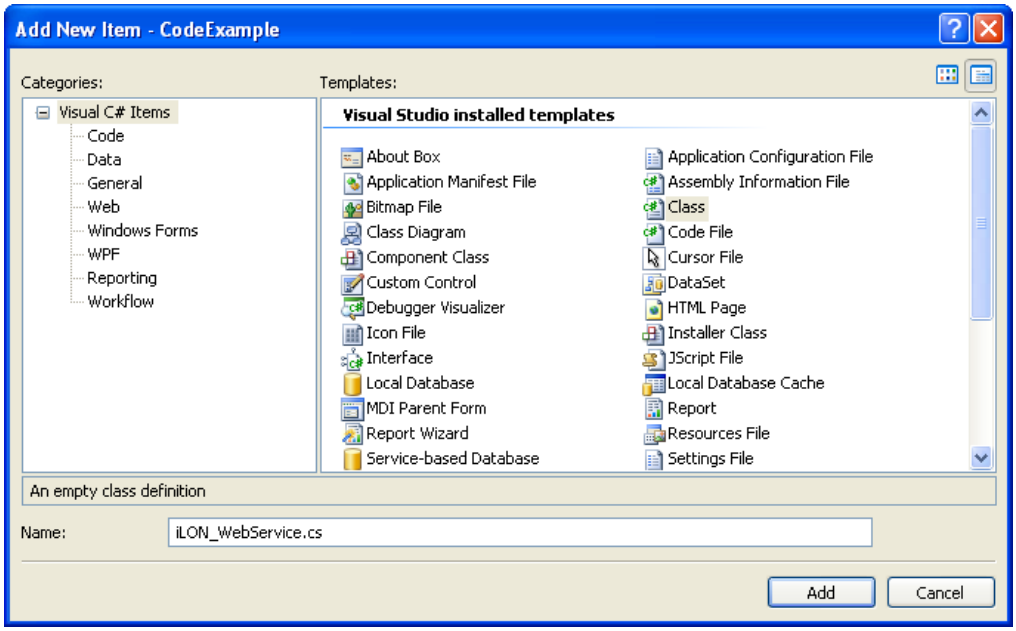


e. Click **Add Reference**. The new Web reference appears in the list of references in the **Solution Explorer** pane.

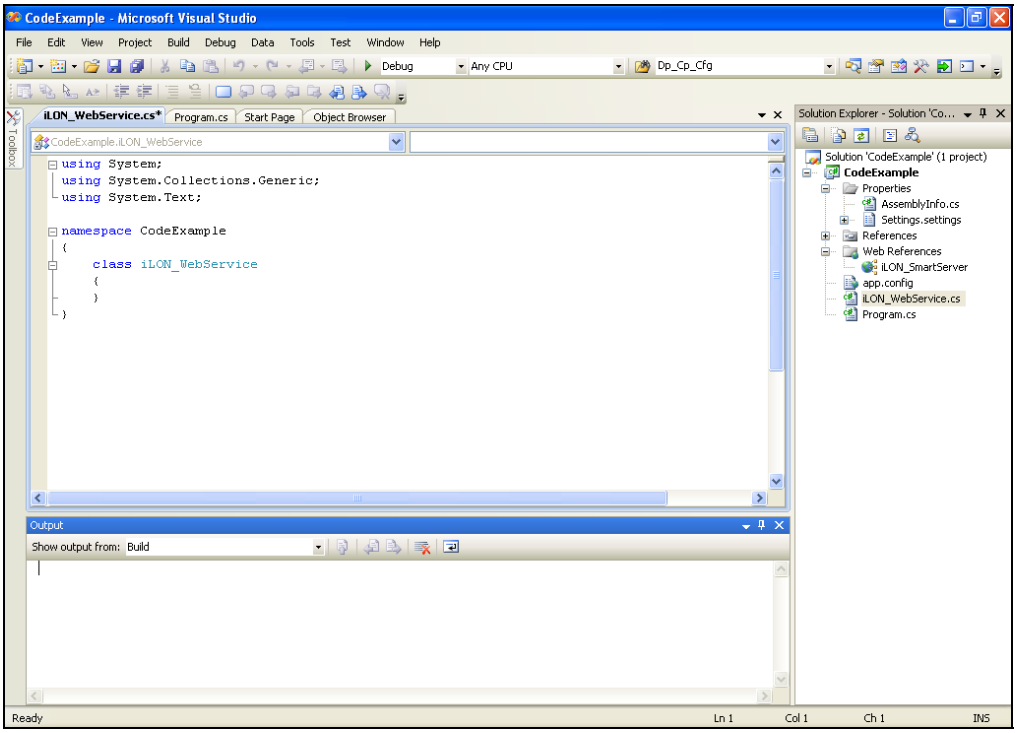


5. A .NET 2.0 client must turn off the keep-alive attribute to communicate with the SmartServer without exceptions being generated. To turn off the keep-alive attribute, the generated web reference class must be inherited, and the *GetWebRequest* method must be overridden, where the **KeepAlive** flag can be turned off. The following procedure describes how to do so.

   a. Click **Project**, and then click **Add Class**. The **Add New Item** dialog opens.



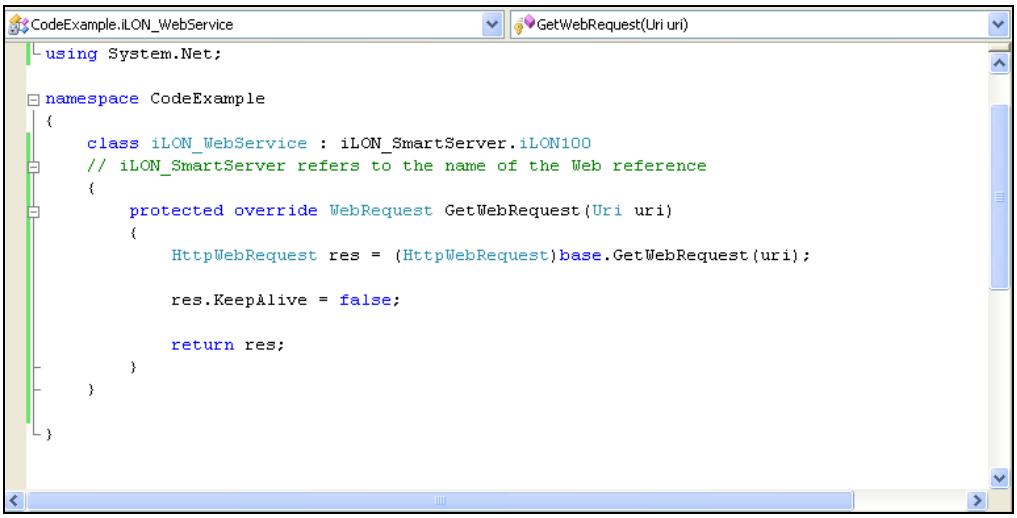
   b. In the **Name** box, enter "iLON_WebService" and then click **Add.**



   c. Insert a **using System.Net** statement at the beginning of the **ilon_WebService** class.

   d. Inside the **iLON_WebService** class, enter the following code to override the *GetWebRequest* function so that it turns off the **KeepAlive** flag:

```
class iLON_WebService : iLON_SmartServer.iLON100

 // iLON_SmartServer refers to the name of the Web reference created in step 4
 {
    protected override WebRequest GetWebRequest(Uri uri)
    {
        HttpWebRequest res = (HttpWebRequest)base.GetWebRequest(uri);
        res.KeepAlive = false;
        return res;
    }
}
```



6. You should now use the **iLON_WebService** class to instantiate the SmartServer Web reference as described in section 20.2.2, *Instantiating the Web Service Client in Visual C# .NET 2.0*.

20.2 Instantiating and Initializing the Web Service Client

Before you can use the functions of the SOAP/XML interface, you must instantiate the Web service object that was referenced in the previous section from within your application. This section contains programming samples written in Visual C# .NET 3.5, Visual C#.NET 2.0, and Visual Basic .NET 3.5 that demonstrate how to do so. For simplicity, the programming samples include all the code required to instantiate the Web service within a single function. This function, for example, could be an event handler for a button click event. You can instantiate the Web service in any routine, although you should generally consider doing this in an initialization routine.

Once you have instantiated the Web service object, you have to set the Web service's URL. This is also known as the *SOAP endpoint*, *EndPointURL*, or *Service endpoint*, depending on which development tool you are using. This is the destination on the SmartServer where SOAP messages from your application will be sent.

In addition, if you have password-protected the WSDL file on the SmartServer with the **i.LON Web Server Security and Parameters** utility, your application needs to specify the correct user ID and password to successfully send SOAP messages to the SmartServer. You can perform this task after you instantiate the Web service, as shown below. For more information on the **i.LON Web Server Security and Parameters** utility, see Appendix C of the *i.LON SmartServer 2.0 User's Guide*.

20.2.1 Instantiating the Web Service Client in Visual C# .NET 3.5

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel; //make sure you add this statement to iLON_SoapCalls class
```

```
namespace SmartServerConsoleExample
{
    class iLON_SoapCalls
    {
        // your SmartServer's IpAddress
        public static string _iLonEndpointIpAddress = "<SmartServer IP address>";

        // your SmartServer's Web service reference
        static public iLON_SmartServer.iLON100portTypeClient _iLON = null;

        /// <summary>
        ///   Instantiates the SmartServer Web service for
        ///   .NET 3.5
        /// </summary>
        static public void BindClientToSmartServer()
        {
            // Specify the binding to be used for the client.
            BasicHttpBinding binding = new BasicHttpBinding();

            // Initialize the namespace
                binding.Namespace = "http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/";

            // Obtain the URL of the Web service on the i.LON SmartServer.
            System.ServiceModel.EndpointAddress endpointAddress
              = new System.ServiceModel.EndpointAddress("http://"
              + _iLonEndpointIpAddress +  "/WSDL/iLON100.wsdl");

            // Instantiate the SmartServer Web service object with this address and binding.
            _iLON = new iLON_SmartServer.iLON100portTypeClient(binding, endpointAddress);

           // Uncommment the lines below to enable authentication
           // binding.Security.Mode =
           // System.ServiceModel.BasicHttpSecurityMode.TransportCredentialOnly;

           // binding.Security.Transport.ClientCredentialType =
           // System.ServiceModel.HttpClientCredentialType.Basic;

           // _iLON.ChannelFactory.Credentials.UserName.UserName = "ilon";
           // _iLON.ChannelFactory.Credentials.UserName.Password = "ilon";
        }


        /// <summary>
        ///   Close the SmartServer Web service
        /// </summary>
        static public void CloseBindingToSmartServer()
        {
            // Closing the client gracefully
            // closes the connection and cleans up resources
            try
            {
                _iLON.Close();
            }
            finally
            {
                _iLON = null;
            }
        }
    }
}
```

20.2.2 Instantiating the Web Service Client in Visual C# .NET 2.0

```
using System;
using System.Collections.Generic;
using System.Text;

namespace CodeExample
{
    class iLON_SoapCalls
    {
        // your SmartServer's Web service reference
        static public iLON_WebService _iLON = null;

        /// <summary>
        ///   Instantiates the SmartServer Web service for .NET 2.0
        /// </summary>
        static public void BindClientToSmartServer(string _iLonEndpointIpAddress)
        {
            _iLON = new iLON_WebService();
            String strOrigUrl = _iLON.Url;
            _iLON.Url = strOrigUrl.Replace("localhost", _iLonEndpointIpAddress);
            _iLON.messagePropertiesValue = new iLON_SmartServer.messageProperties();

            // uncomment the 2 lines below to enable authentication
            //            _iLON.Credentials = new System.Net.NetworkCredential("ilon", "ilon");
            //            _iLON.PreAuthenticate = true;
        }

    }
}
```

20.2.3 Instantiating the Web Service Client in Visual Basic .NET 3.5

The following example shows how to instantiate the Web service in Visual Basic. NET:

```
Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text
Imports System.ServiceModel 'make sure you add this statement to iLON_SoapCalls class

Public Class iLON_SoapCalls

    'your SmartServer's IpAddress
    Public _iLonEndpointIpAddress As String = "<SmartServer IP address>"

    'your SmartServer's Web service reference
    Public _iLON As iLON_SmartServer.iLON100portTypeClient = Nothing

    ''' <summary>
    ''' Instantiate the SmartServer Web service for .NET 3.0 and 3.5 (NOT 2.0)
    ''' </summary>

    Public Sub BindClientToSmartServer()

        ' Specify the binding to be used for the client.
        Dim binding As BasicHttpBinding = New BasicHttpBinding()

        ' Initialize the namespace
        binding.Namespace = "http:'wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/"

        ' Obtain the URL of the Web service on the i.LON SmartServer.
        Dim endpointAddress As System.ServiceModel.EndpointAddress =
           New System.ServiceModel.EndpointAddress("http://" + _iLonEndpointIpAddress + "/WSDL/iLON100.wsdl")

        ' Instantiate the SmartServer Web service object with this address and binding.
        _iLON = New iLON_SmartServer.iLON100portTypeClient(binding, endpointAddress)

        ' uncomment the lines below to enable authentication
        ' binding.Security.Mode = System.ServiceModel.BasicHttpSecurityMode.TransportCredentialOnly
        ' binding.Security.Transport.ClientCredentialType = System.ServiceModel.HttpClientCredentialType.Basic
        ' _iLON.ChannelFactory.Credentials.UserName.UserName = "ilon"
        ' _iLON.ChannelFactory.Credentials.UserName.Password = "ilon"

    End Sub

    ''' <summary>
    '''   Close the SmartServer Web service
    ''' </summary>

    Public Sub CloseBindingToSmartServer()
        ' Closing the client gracefully
        ' closes the connection and cleans up resources
        Try
            _iLON.Close()
        Finally
            _iLON = Nothing
        End Try

    End Sub

End Class
```

20.3 Calling Web Services Methods

The following examples demonstrate how to read and write values to a data point in Visual C# .NET 3.5, Visual C#.NET 2.0, and Visual Basic .NET 3.5, and how to setup a Web connection between a SmartServer and a WebBinder Target Server.

Note: The following examples assume that you are using a SmartServer that has been set to its factory default settings. This prevents compilation errors based on mismatching <UCPTname> properties of the objects in the LONWORKS network hierarchy (*network*/*channel*/*device*/*functional block*/*data point*).

20.3.1 Reading and Writing Data Point Values in Visual C# .NET 3.5

The following Visual C# .NET 3.5 example reads the value of the Net/LON/iLON App/Digital Output 1/nviClaValue_1 data point on the SmartServer, and then writes a value of "100.0 1" to it. This **SNVT_switch** data point is one of the relay outputs on the SmartServer. You can execute this code after you have completed section 20.1.1, *Referencing and Inheriting from the WSDL Using .NET 3.5 Framework*, and section 20.2.1, *Instantiating the Web Service Client in Visual C# .NET 3.5*.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SmartServerConsoleExample
{
class Program
{
    static void Main(string[] args)
    {
       iLON_SoapCalls.BindClientToSmartServer();
       iLON_SmartServer.iLON100portTypeClient SmartServer = iLON_SoapCalls._iLON;


       // -------------- READING A DATA POINT VALUE --------------

      try
      {
       // instantiate the member object
        iLON_SmartServer.Item_Coll itemColl = new iLON_SmartServer.Item_Coll();
        itemColl.Item = new iLON_SmartServer.Item[1];
        itemColl.Item[0] = new iLON_SmartServer.Dp_Data();

        // set the DP name
        itemColl.Item[0].UCPTname = "Net/LON/iLON App/Digital Output 1/nviClaValue_1";

        // set maxAge to get the updated DP value in case it has been cached for more than 10
        // seconds on the Data Server (see section 4.3.4.1 for more information)
        ((iLON_SmartServer.Dp_Data)(itemColl.Item[0])).UCPTmaxAge = 10;
        ((iLON_SmartServer.Dp_Data)(itemColl.Item[0])).UCPTmaxAgeSpecified = true;

        //call the Read Function
        iLON_SmartServer.Item_DataColl dataColl = SmartServer.Read(itemColl);

        if (dataColl.Item == null)
        {
            // sanity check.  this should not happen
            Console.Out.WriteLine("No items were returned");
        }

        else if (dataColl.Item[0].fault != null)
        {
            // error
            Console.Out.WriteLine("An error occurred.  Fault code = " +
            dataColl.Item[0].fault.faultcode +
            ".  Fault text = %s." +
            dataColl.Item[0].fault.faultstring);
        }

        else
        {
            // success
            Console.Out.WriteLine("Read is successful");
            Console.Out.WriteLine(((iLON_SmartServer.Dp_Data)dataColl.Item[0]).UCPTname + " = " +
           ((iLON_SmartServer.Dp_Data)dataColl.Item[0]).UCPTvalue[0].Value + "\n");
        }
```

```
        // -------------- WRITING A DATA POINT VALUE --------------

        // reset the DP priority (see section 4.3.6 for more information)
        iLON_SmartServer.Item_Coll itemCollInvoke = new iLON_SmartServer.Item_Coll();
        itemCollInvoke.Item = new iLON_SmartServer.Item[1];
        itemCollInvoke.Item[0] = new iLON_SmartServer.Dp_ResetPrio_Invoke();

        ((iLON_SmartServer.Dp_ResetPrio_Invoke)(itemCollInvoke.Item[0])).UCPTname =
         "Net/LON/iLON App/Digital Output 1/nviClaValue_1";
        ((iLON_SmartServer.Dp_ResetPrio_Invoke)(itemCollInvoke.Item[0])).UCPTpriority = 200;
        ((iLON_SmartServer.Dp_ResetPrio_Invoke)(itemCollInvoke.Item[0])).UCPTprioritySpecified = true;
        SmartServer.InvokeCmd(ref itemCollInvoke);

        // set the DP priority to 200 (see section 4.3.7 for more information)
        ((iLON_SmartServer.Dp_Data)dataColl.Item[0]).UCPTpriority = 200;
        ((iLON_SmartServer.Dp_Data)dataColl.Item[0]).UCPTprioritySpecified = true;

        // set 100.0 1 as the value
        ((iLON_SmartServer.Dp_Data)dataColl.Item[0]).UCPTvalue = new iLON_SmartServer.E_LonString[1];
        ((iLON_SmartServer.Dp_Data)dataColl.Item[0]).UCPTvalue[0] = new iLON_SmartServer.E_LonString();
        ((iLON_SmartServer.Dp_Data)dataColl.Item[0]).UCPTvalue[0].Value = "100.0 1";

        // to write a preset, do this (see section 4.3.5.2 for more information)
        // dpData.UCPTvalue[0].LonFormat = "UCPTvalueDef";
        // dpData.UCPTvalue[0].Value = "ON";

        // call the write function
        iLON_SmartServer.Item_Coll writeResp = SmartServer.Write(dataColl);

        if (writeResp.Item == null)
        {
            // sanity check.  this should not happen
            Console.Out.WriteLine("No items were returned");
        }

        else if (writeResp.Item[0].fault != null)
        {
            // error
            Console.Out.WriteLine("An error occurred.  Fault code = " +
                writeResp.Item[0].fault.faultcode +
                                ".  Fault text = %s." +
                                writeResp.Item[0].fault.faultstring);
        }

        else
        {
            // success
            Console.Out.WriteLine("Write is successful");
            Console.Out.WriteLine(((iLON_SmartServer.Dp_Data)dataColl.Item[0]).UCPTname + " = " +
            ((iLON_SmartServer.Dp_Data)dataColl.Item[0]).UCPTvalue[0].Value);
        }

       Console.ReadLine();
      }

     finally
     {
        iLON_SoapCalls.CloseBindingToSmartServer();
     }
  }

 }
}
```

20.3.2 Reading and Writing Data Point Values in Visual C# .NET 2.0

The following Visual C# .NET 2.0 example reads the value of the Net/LON/iLON App/Digital Output 1/nviClaValue_1 data point on the SmartServer, and then writes a value of “100.0 1” to it. This **SNVT_switch** data point is one of the relay outputs on the SmartServer. You can execute this code

after you have completed section 20.2.1, *Referencing and Inheriting from the WSDL Using .NET 2.0 Framework*, and section 20.2.2, *Instantiating the Web Service Client in Visual C# .NET 2.0*.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SmartServerConsoleExample
{
    class Program
    {
        // your SmartServer's IpAddress
        public static string _iLonEndpointIpAddress = "<SmartServer IP Address>";

        static void Main(string[] args)
        {
            iLON_SoapCalls.BindClientToSmartServer(_iLonEndpointIpAddress);
            iLON_SmartServer.iLON100 SmartServer = iLON_SoapCalls._iLON;

            // -------------- READ DATA POINT VALUE --------------

          try
          {
            // instantiate the member object
            iLON_SmartServer.Item_Coll itemColl = new iLON_SmartServer.Item_Coll();
            itemColl.Item = new iLON_SmartServer.Item[1];
            itemColl.Item[0] = new iLON_SmartServer.Dp_Data();

            // set the DP name
            itemColl.Item[0].UCPTname = "Net/LON/iLON App/Digital Output 1/nviClaValue_1";

            // set maxAge to get the updated DP value in case it has been cached for more than 10
            // seconds on the Data Server (see section 4.3.4.1 for more information)
            ((iLON_SmartServer.Dp_Data)(itemColl.Item[0])).UCPTmaxAge = 10;
            ((iLON_SmartServer.Dp_Data)(itemColl.Item[0])).UCPTmaxAgeSpecified = true;

            //call the Read Function
            iLON_SmartServer.Item_DataColl dataColl = SmartServer.Read(itemColl);

            if (dataColl.Item == null)
            {
                // sanity check.  this should not happen
                Console.Out.WriteLine("No items were returned");
            }

            else if (dataColl.Item[0].fault != null)
            {
                // error
                Console.Out.WriteLine("An error occurred.  Fault code = " +
                dataColl.Item[0].fault.faultcode +
                ".  Fault text = %s." +
                dataColl.Item[0].fault.faultstring);
            }

            else
            {
                // success
                Console.Out.WriteLine("Read is successful");
                Console.Out.WriteLine(((iLON_SmartServer.Dp_Data)dataColl.Item[0]).UCPTname + " = " +
               ((iLON_SmartServer.Dp_Data)dataColl.Item[0]).UCPTvalue[0].Value + "\n");
            }

            // -------------- WRITE DATA POINT VALUE --------------

           // reset the DP priority (see section 4.3.6 for more information)
            iLON_SmartServer.Item_Coll itemCollInvoke = new iLON_SmartServer.Item_Coll();
            itemCollInvoke.Item = new iLON_SmartServer.Item[1];
            itemCollInvoke.Item[0] = new iLON_SmartServer.Dp_ResetPrio_Invoke();

            ((iLON_SmartServer.Dp_ResetPrio_Invoke)(itemCollInvoke.Item[0])).UCPTname =
```

```
            "Net/LON/iLON App/Digital Output 1/nviClaValue_1";
           ((iLON_SmartServer.Dp_ResetPrio_Invoke)(itemCollInvoke.Item[0])).UCPTpriority = 200;
           ((iLON_SmartServer.Dp_ResetPrio_Invoke)(itemCollInvoke.Item[0])).UCPTprioritySpecified = true;
           SmartServer.InvokeCmd(ref itemCollInvoke);

           // set the DP priority to 200 (see section 4.3.7 for more information)
           ((iLON_SmartServer.Dp_Data)dataColl.Item[0]).UCPTpriority = 200;
           ((iLON_SmartServer.Dp_Data)dataColl.Item[0]).UCPTprioritySpecified = true;

           // set 100.0 1 as the value
          ((iLON_SmartServer.Dp_Data)dataColl.Item[0]).UCPTvalue = new iLON_SmartServer.E_LonString[1];
           ((iLON_SmartServer.Dp_Data)dataColl.Item[0]).UCPTvalue[0] = new iLON_SmartServer.E_LonString();
           ((iLON_SmartServer.Dp_Data)dataColl.Item[0]).UCPTvalue[0].Value = "100.0 1";

           // to write a preset, do this (see section 4.3.5.2 for more information)
           // dpData.UCPTvalue[0].LonFormat = "UCPTvalueDef";
           // dpData.UCPTvalue[0].Value = "ON";

           // call the write function
           iLON_SmartServer.Item_Coll writeResp = SmartServer.Write(dataColl);

           if (writeResp.Item == null)
           {
               // sanity check.  this should not happen
               Console.Out.WriteLine("No items were returned");
           }

           else if (writeResp.Item[0].fault != null)
           {
               // error
               Console.Out.WriteLine("An error occurred.  Fault code = " +
                   writeResp.Item[0].fault.faultcode +
                                   ".  Fault text = %s." +
                                  writeResp.Item[0].fault.faultstring);
           }

           else
           {
               // success
               Console.Out.WriteLine("Write is successful");
               Console.Out.WriteLine(((iLON_SmartServer.Dp_Data)dataColl.Item[0]).UCPTname + " = " +
               ((iLON_SmartServer.Dp_Data)dataColl.Item[0]).UCPTvalue[0].Value);
           }

       Console.ReadLine();
     }

    finally
    {
       iLON_SoapCalls.CloseBindingToSmartServer();
    }
  }
 }
}
```

20.3.3 Reading and Writing Data Point Values in Visual Basic .NET 3.5

The following Visual Basic .NET 3.5 example reads the value of the Net/LON/iLON App/Digital Output 1/nviClaValue_1 data point on the SmartServer, and then writes a value of "100.0 1" to it. This **SNVT_switch** data point is one of the relay outputs on the SmartServer. You can execute this code after you have completed section 20.1.1, *Referencing and Inheriting from the WSDL Using .NET 3.5 Framework*, and section 20.2.3,*Instantiating the Web Service Client in Visual Basic .NET 3.5*.

```
Module Module1

    Sub Main()

        Dim SmartServer As iLON_SoapCalls = New iLON_SoapCalls

       // Enter your SmartServer's IpAddress
        SmartServer.BindClientToSmartServer("<SmartServer IP Address>")

        Try

        ' -------------- READING A DATA POINT VALUE --------------

        ' instantiate the member object
        Dim itemColl As New iLON_SmartServer.Item_Coll()
        itemColl.Item = New iLON_SmartServer.Item(0) {}
        itemColl.Item(0) = New iLON_SmartServer.Dp_Data()

        ' set the DP name
        itemColl.Item(0).UCPTname = "Net/LON/iLON App/Digital Output 1/nviClaValue_1"

        ' set maxAge to get the updated DP value in case it has been cached for more than 10
        ' seconds on the Data Server (see section 4.3.4.1 for more information)
        DirectCast((itemColl.Item(0)), iLON_SmartServer.Dp_Data).UCPTmaxAge = 10
        DirectCast((itemColl.Item(0)), iLON_SmartServer.Dp_Data).UCPTmaxAgeSpecified = True


        'call the Read Function
        Dim dataColl As iLON_SmartServer.Item_DataColl = SmartServer._iLON.Read(itemColl)

        If dataColl.Item Is Nothing Then

            ' sanity check.  this should not happen
            Console.Out.WriteLine("No items were returned")

        ElseIf dataColl.Item(0).fault IsNot Nothing Then

            ' error
           Console.Out.WriteLine(("An error occurred.  Fault code = " +
           dataColl.Item(0).fault.faultcode.Value + ".  Fault text = %s.") +
           dataColl.Item(0).fault.faultstring)
        Else

            ' success
            Console.Out.WriteLine("Read is successful")
            Console.Out.WriteLine((DirectCast(dataColl.Item(0), iLON_SmartServer.Dp_Data).UCPTname &
              " = ") + DirectCast(dataColl.Item(0), iLON_SmartServer.Dp_Data).UCPTvalue(0).Value &
               vbLf)
        End If

        ' -------------- WRITING A DATA POINT VALUE --------------

       ' reset the DP priority (see section 4.3.6 for more information)
        Dim itemCollInvoke As New iLON_SmartServer.Item_Coll()
        itemCollInvoke.Item = New iLON_SmartServer.Item(0) {}
        itemCollInvoke.Item(0) = New iLON_SmartServer.Dp_ResetPrio_Invoke()

         DirectCast((itemCollInvoke.Item(0)), iLON_SmartServer.Dp_ResetPrio_Invoke).UCPTname =
       "Net/LON/iLON App/Digital Output 1/nviClaValue_1"
        DirectCast((itemCollInvoke.Item(0)), iLON_SmartServer.Dp_ResetPrio_Invoke).UCPTpriority = 200
        DirectCast((itemCollInvoke.Item(0)), iLON_SmartServer.Dp_ResetPrio_Invoke).UCPTprioritySpecified = True
```

```
        SmartServer._iLON.InvokeCmd(itemCollInvoke)

        ' set the DP priority to 200 (see section 4.3.7 for more information)
        DirectCast(dataColl.Item(0), iLON_SmartServer.Dp_Data).UCPTpriority = 200
        DirectCast(dataColl.Item(0), iLON_SmartServer.Dp_Data).UCPTprioritySpecified = True

        ' set 100.0 1 as the value
        DirectCast(dataColl.Item(0), iLON_SmartServer.Dp_Data).UCPTvalue = New iLON_SmartServer.E_LonString(0) {}
        DirectCast(dataColl.Item(0), iLON_SmartServer.Dp_Data).UCPTvalue(0) = New iLON_SmartServer.E_LonString()
        DirectCast(dataColl.Item(0), iLON_SmartServer.Dp_Data).UCPTvalue(0).Value = "100.0 1"

       ' to write a preset, do this (see section 4.3.5.2 for more information)
        ' DirectCast(dataColl.Item(0), iLON_SmartServer.Dp_Data).UCPTvalue(0).LonFormat = "UCPTvalueDef"
        ' DirectCast(dataColl.Item(0), iLON_SmartServer.Dp_Data).UCPTvalue(0).Value = "ON"

        ' call the write function
        Dim writeResp As iLON_SmartServer.Item_Coll = SmartServer._iLON.Write(dataColl)

        If writeResp.Item Is Nothing Then
            ' sanity check.  this should not happen
            Console.Out.WriteLine("No items were returned")

        ElseIf writeResp.Item(0).fault IsNot Nothing Then
            ' error
               Console.Out.WriteLine(("An error occurred.  Fault code = " +
                 writeResp.Item(0).fault.faultcode.Value + ".  Fault text = %s.") +
                 writeResp.Item(0).fault.faultstring)
        Else

            ' success
            Console.Out.WriteLine("Write is successful")
            Console.Out.WriteLine((DirectCast(dataColl.Item(0), iLON_SmartServer.Dp_Data).UCPTname
              & " = ") + DirectCast(dataColl.Item(0), iLON_SmartServer.Dp_Data).UCPTvalue(0).Value)
        End If

        Console.ReadLine()

       Finally

        SmartServer.CloseBindingToSmartServer()

        End Try

    End Sub

End Module
```

20.4 Accepting a Web Binding From a SmartServer

To create a Web connection between the SmartServer and your Web server, you need to expose a Web service on your server. This section describes how to do so with Microsoft Visual Studio 2008 and .NET Framework 3.5. You need to configure IIS (Web server) on your computer so that it can serve the Web service that you are going to write in the following section. This section assumes you are familiar with IIS configuration and Web server administration.

Notes:

- In order for your .NET application to support Web connection file attachment, you must download the **Web Services Enhancements 2.0 Add-On** from Microsoft's Web site at *msdn.microsoft.com* and install it on your computer.
- If you are not running the .NET Framework 3.5, you can download the wsdl.exe file from Microsoft's Web site at *msdn.microsoft.com* to use this example with another development environment.

To create a Web Binding, follow these steps:

1. Create a proxy class with the wsdl.exe Web services description language tool. To do this follow these steps:

a. Open a Command Prompt window to the following folder on your computer:

   C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin folder.

b. Enter the following command (in one line):

   wsdl.exe /language:CS /serverInterface /protocol:SOAP /n:iLon100e4 /u:<*SmartServer user name*> /p:< *SmartServer password*> "http://<*SmartServer IP address*>/WSDL/V4.0/iLON100.wsdl" /out: "<*output directory*>\iLON100proxy.cs"

   SmartServer user name is the name used to log on to your SmartServer. This is **ilon** by default.

   SmartServer password is the password used to log on to your SmartServer. This is **ilon** by default.

   SmartServer IP address is the IP address of your SmartServer.

   output directory is the destination folder on your computer where the proxy class is to be stored.

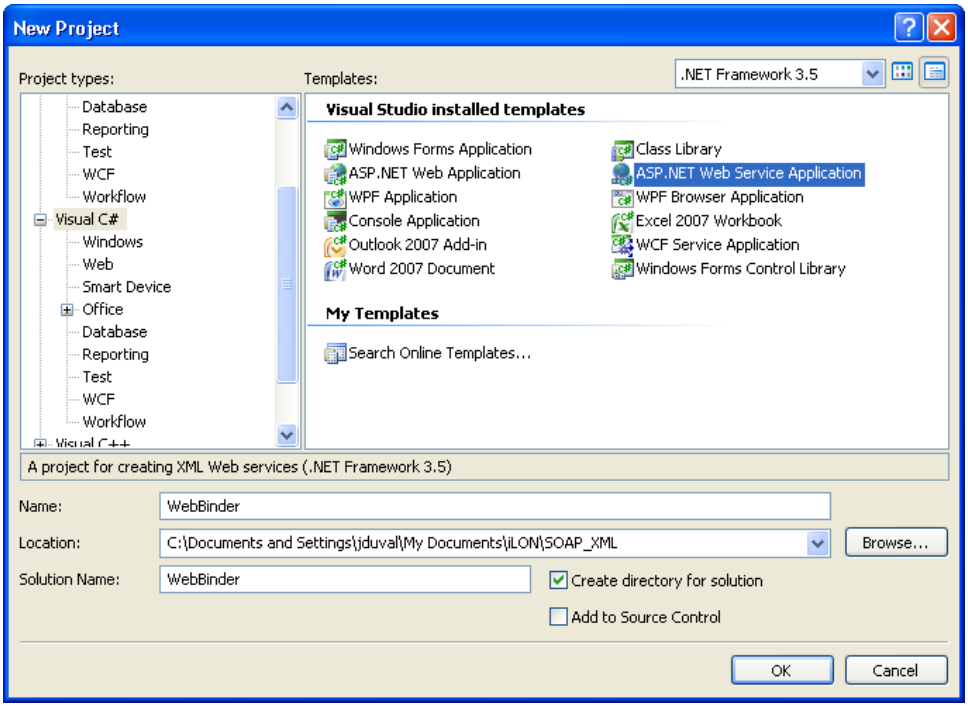
   The following example demonstrates how to enter a command that stores the proxy class in a C:/LonWorks/SmartServerProxyClass folder:

   wsdl.exe /language:CS /serverInterface /protocol:SOAP /n:iLon100e4 /u:ilon /p:ilon "http://10.2.124.82/WSDL/V4.0/iLON100.wsdl" /out:"C:\lonworks\SmartServerProxyClass\iLON100proxy.cs"

   Note: /serverInterface is used instead of /server because the /server Server switch has been deprecated. Using the /serverInterface switch generates an abstract class for an XML web service implementation using ASP.NET based on the contracts. The default is to generate a client proxy class.

2. A file called **iLON100.cs** is generated in the specified destination folder. You will use this file after you create a new Web service project. You can optionally specify other languages such as Visual Basic .NET. See the MSDN documentation for more information on this command.

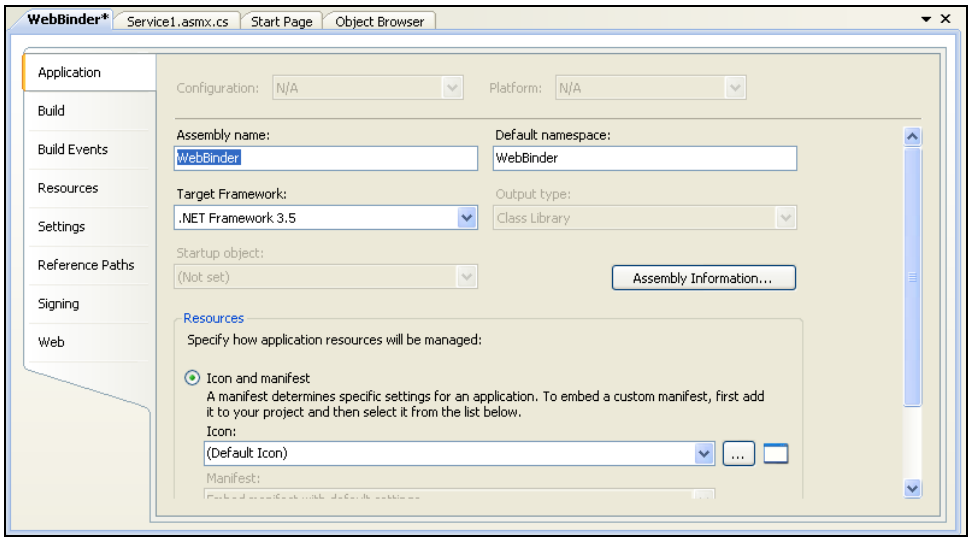
3. Create a new Web service project using **ASP .NET Web Service**. The sample code below is written in Visual C# .NET, and uses "WebBinder" as the project name.



4. Add a reference to the **Microsoft.Web.Services2.dll** component. To do this, click **Project** and then select **Add References**. The **Add References** dialog opens. Click the **Browse** tab, browse to the C:\Program Files\Microsoft WSE\v2.0 folder on your computer, click the **Microsoft.Web.Services2.dll** file, and then click **OK**.

5. Add the **iLON100.cs** proxy class to the project (you can copy the file to the same folder used to store your source code). To add the proxy class, click **Project**, click **Add Existing Item**, browse to the folder where the **iLON100.cs** file is stored, and then select the **iLON100.cs** file. This allows you to use the complex soap types that the *Write* function uses on the data points on the Data Server.

6. Optionally, you can change the target framework to .NET Framework 3.5. Note that the IIS ASP.NET 2.0x version will still be used because the ASP.NET used by .NET Framework 3.5 is simply an extension of the 2.0x version. To do this, click **Project** and then click **WebBinder Properties**. In the **Target Framework** box of the **WebBinder** tab, select .NET Framework 3.5.



7. Write the code for web service. You can simply copy and paste the following code snippet into the public class **Service1 : System.Web.Services.WebService**.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Linq;
using iLon100e4;
using Microsoft.Web.Services2;

namespace WebBinder
{
    /// <summary>
    /// Summary description for Service1
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [ToolboxItem(false)]
    // To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the
    //following line:
    // [System.Web.Script.Services.ScriptService]

    public class Service1 : System.Web.Services.WebService, IILON100soap11Binding {
        private messageProperties _messagePropertiesValue = null;
        public messageProperties messagePropertiesValue {
            get;
            set;
        }


        /// <remarks/>
        public Item_Coll List(E_xSelect iLonItem) {
            return null;
        }

        /// <remarks/>
        public Item_CfgColl Get(Item_Coll iLonItem) {
            return null;
        }

        /// <remarks/>
        public Item_Coll Set(Item_CfgColl iLonItem) {
            return null;
        }

        /// <remarks/>
        public void Delete(ref Item_Coll iLonItem) {
        }

        /// <remarks/>
        public Item_DataColl Read(Item_Coll iLonItem) {
            return null;
        }

        /// <remarks/>
        public Item_Coll Write(Item_DataColl iLonItem) {
           System.Diagnostics.Trace.WriteLine("Got message from : " +
           messagePropertiesValue.UCPTipAddress);

            #region just for debugging, not needed as system throws exceptions anyway
            // are the expected object existing ?
            if ((null == iLonItem) || (null == iLonItem.Item[0])) {
                System.Diagnostics.Trace.WriteLine("ERROR, Null object");
                throw new System.NullReferenceException();
            }
            #endregion
```

```
            // create the response object
            Item_Coll itemColl_resp = new Item_Coll();
            itemColl_resp.Item = new Item[] { new Item() };
            itemColl_resp.Item[0].UCPTname = iLonItem.Item[0].UCPTname;

            // is the Item of the expected type?
            if (!(iLonItem.Item[0] is Dp_Data)) {
                System.Diagnostics.Trace.WriteLine("ERROR, unexpected object type");
                ++itemColl_resp.UCPTfaultCount;
                itemColl_resp.UCPTfaultCountSpecified = true;

                itemColl_resp.Item[0].fault = new E_Fault();
                itemColl_resp.Item[0].fault.faultcode = new E_FaultFaultcode();
                itemColl_resp.Item[0].fault.faultcode.faultType = Fault_eType._error;
                itemColl_resp.Item[0].fault.faultcode.Value = 12; /* eFECommandFailed */
                itemColl_resp.Item[0].fault.faultstring = "Unexpected object type";
            }
            // everything is fine, proceed..
            else {
                Dp_Data dpData = (Dp_Data)iLonItem.Item[0];
                System.Diagnostics.Trace.WriteLine(String.Format("The value is: '{0}'",
                dpData.UCPTvalue[0].Value));

                // Handle the attachment file
                SoapContext soapContext = RequestSoapContext.Current;

                if (soapContext != null) {
                    // If there is an attachment file
                    if (soapContext.Attachments.Count > 0) {
                        System.Diagnostics.Trace.WriteLine("attachment-id: " +
                         soapContext.Attachments[0].Id);
                        if (soapContext.Attachments[0].ContentType == "text/plain"
                         || soapContext.Attachments[0].ContentType == "text/xml")
                        {
                          string attachment;

                            System.IO.StreamReader attachmentStream
                             = new System.IO.StreamReader(soapContext.Attachments[0].Stream);

                            // Read the attachment file to the end
                            attachment = attachmentStream.ReadToEnd();
                            attachmentStream.Close();

                            // Write the contents of the file
                            System.Diagnostics.Trace.WriteLine(attachment);
                        }
                    }
                }
                return itemColl_resp;
            }

            /// <remarks/>
            public void Clear(ref Item_Coll iLonItem) {
            }

            /// <remarks/>
            public void InvokeCmd(ref Item_Coll iLonItem) {
            }
        }
    }
```

8. Before you run the application, you need to change **web.config** file as follows. You can open web.config file from the Solution Explorer. Add the following snippet at the top of the **<configuration>** element.

```
<?xml version="1.0"?>
 <configuration>
    <configSections>
```

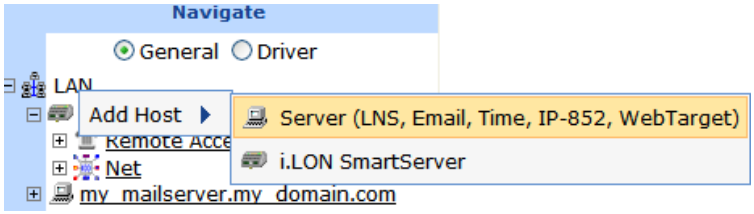
```
        <section name="microsoft.web.services2"
type="Microsoft.Web.Services2.Configuration.WebServicesConfiguration, Microsoft.Web.Services2,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />

        <sectionGroup name="system.web.extensions"
type="System.Web.Configuration.SystemWebExtensionsSectionGroup, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35">

        <sectionGroup name="scripting" type="System.Web.Configuration.ScriptingSectionGroup,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35">

            <section name="scriptResourceHandler"
type="System.Web.Configuration.ScriptingScriptResourceHandlerSection, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" requirePermission="false"
allowDefinition="MachineToApplication"/>

        <sectionGroup name="webServices"
type="System.Web.Configuration.ScriptingWebServicesSectionGroup, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35">

            <section name="jsonSerialization"
type="System.Web.Configuration.ScriptingJsonSerializationSection, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" requirePermission="false"
allowDefinition="Everywhere"/>

            <section name="profileService"
type="System.Web.Configuration.ScriptingProfileServiceSection, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" requirePermission="false"
allowDefinition="MachineToApplication"/>

            <section name="authenticationService"
type="System.Web.Configuration.ScriptingAuthenticationServiceSection, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" requirePermission="false"
allowDefinition="MachineToApplication"/>

            <section name="roleService"
type="System.Web.Configuration.ScriptingRoleServiceSection, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" requirePermission="false"
allowDefinition="MachineToApplication"/>

        </sectionGroup>
        </sectionGroup>
        </sectionGroup>
    </configSections>

  <microsoft.web.services2>
    <diagnostics>
      <detailedErrors enabled="false" />
    </diagnostics>
  </microsoft.web.services2>
```

9. Add the following code under the <**system.web**> tag:

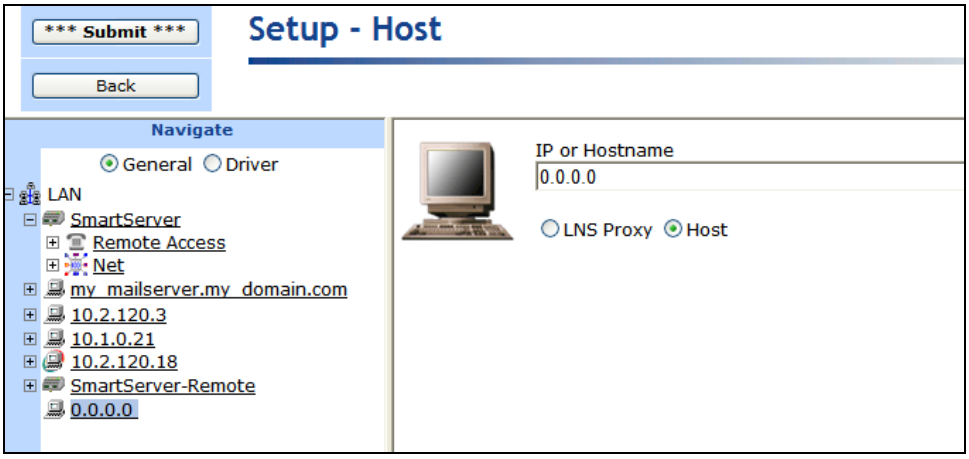
```
  <appSettings/>
    <connectionStrings/>
    <system.web>
    <webServices>
      <soapExtensionTypes>
        <add type="Microsoft.Web.Services2.WebServicesExtension, Microsoft.Web.Services2,
        Version=2.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" priority="1" group="0"
        />
      </soapExtensionTypes>
    </webServices>
```

10. Your server side code is now ready to accept WebBinder calls from the SmartServer, and you can now add your server as a host device to the SmartServer's LAN connection. To do this, follow these steps:

a. Right-click the **LAN** icon or a dial-out connection icon, point to **Add Host**, and then click **Server (LNS, E-mail, Time, IP 852 Config, WebTarget)** on the shortcut menu, or if are you adding the WebBinder Target to an existing server on the LAN, skip to step 4.

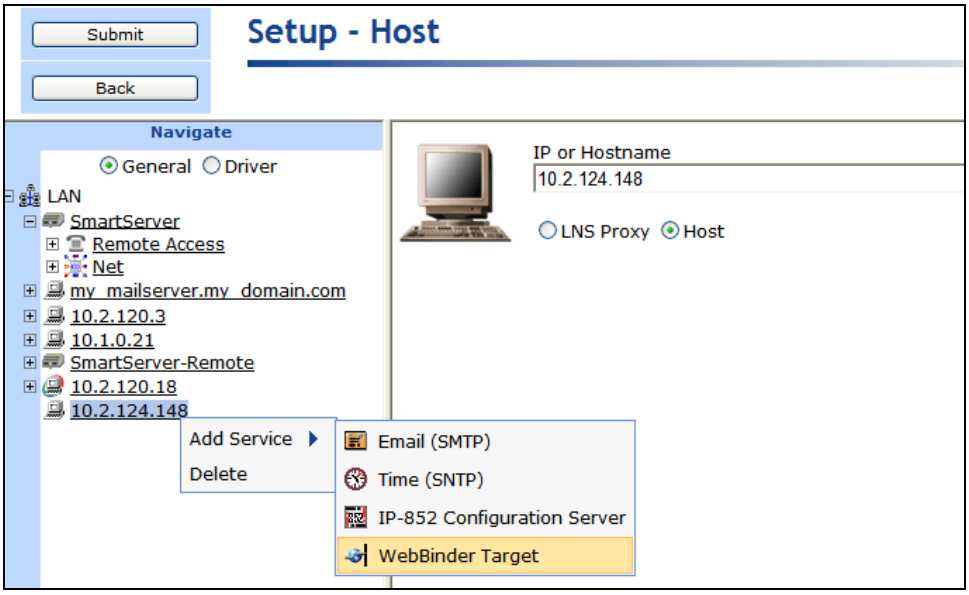


b. The **Setup – Host** Web page opens, and a server icon is added one level below the LAN icon at the bottom of the navigation pane or one level below the dial-out connection icon.

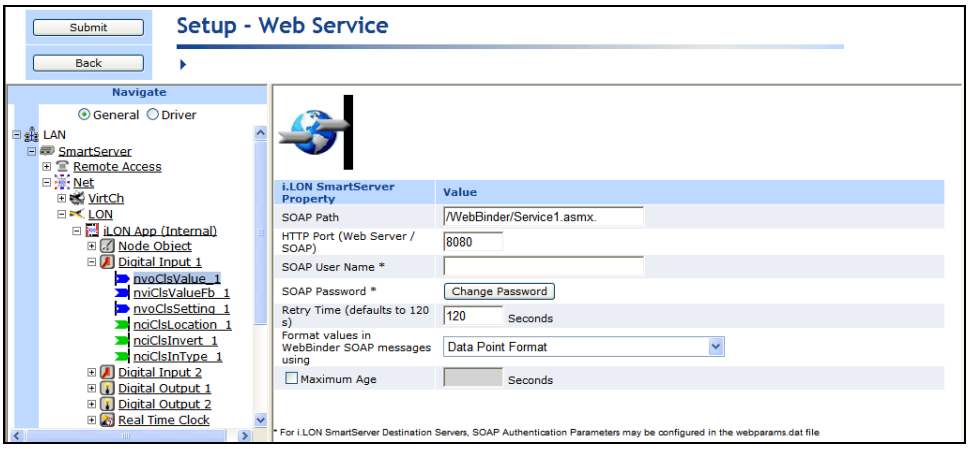


c. Enter the IP address or hostname of the WebBinder Target server and then click **Submit**. The server icon in the tree is updated with the IP address or hostname you entered.

d. Right-click the server icon, point to **Add Service**, then and click **WebBinder Target** on the shortcut menu.



e. The **Setup – Web Service** Web page opens.



f. Configure the following properties for the WebBinder Target server:

i.LON SmartServer Property	
SOAP Path	Enter the path on the WebBinder Target server to which SOAP messages should be transmitted. This is typically the location of the WSDL or ASMX file on the WebBinder target where it receives SOAP messages. For example, if you are exposing your Web service with the namespace of "WebBinder/Service1.asmx", then enter the following text in the **SOAP Path** box: **/WebBinder/Service1.asmx**. **Note**: You must include the leading "/" in the SOAP Path.
HTTP Port (Web Server/SOAP)	Enter the port that the WebBinder Target server uses to serve HTTP requests (SOAP and WebDAV). The default value is **80**, but you may change it to any valid port number. Contact your IS department to ensure your firewall is configured to allow access to the server on this port.
SOAP User Name	Optionally, you can enter a user name to be used for logging in to the WebBinder Target server.
SOAP Password	If you create a user name, click **Change Password** to enter the password to be used for logging in to the WebBinder Target server.
Retry Time	Set the amount of time (in seconds) after which the SmartServer will stop attempting to resend failed WebBinder connection messages to the WebBinder Target server. The default value is **120** seconds. The SmartServer automatically attempts to resend failed WebBinder connection messages every 45 seconds.
Format Values in WebBinder SOAP Messages Using	Select how data point values are formatted in SOAP messages sent to this WebBinder Target server via Web connections. You have two choices: • **Data Point Format**. Data point values are formatted based on the SNVT, UNVT, SCPT, or UCPT defined for the data point. • **Raw HEX**. Data point values are transmitted in raw

hexadecimal format.

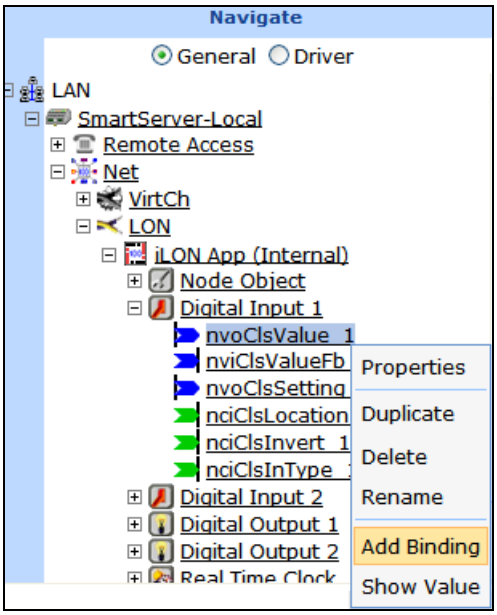
Maximum Age	Specify the maximum age (in seconds) to be written to the target data points on the WebBinder destination when the local SmartServer sends updated values to them. If the WebBinder destination cannot communicate with the parent device of the target data point, the WebBinder destination caches the updated value it received from the local SmartServer. When the device goes online, the cached value is written to the target data point provided that time the value has been cached is less than the maximum age. If the value has been cached longer than the maximum age, the value is not written to the target data point.

g. Click **Submit** to save the changes.

11. You can open a Web browser and enter the IP address of your Web service, such as http://192.168.1.100/WebBinder/Service1.asmx. This lets you test the Web page for Service1, where the *Write* function is the available Web service. The SmartServer will consume this Web service when it makes WebBinder calls.

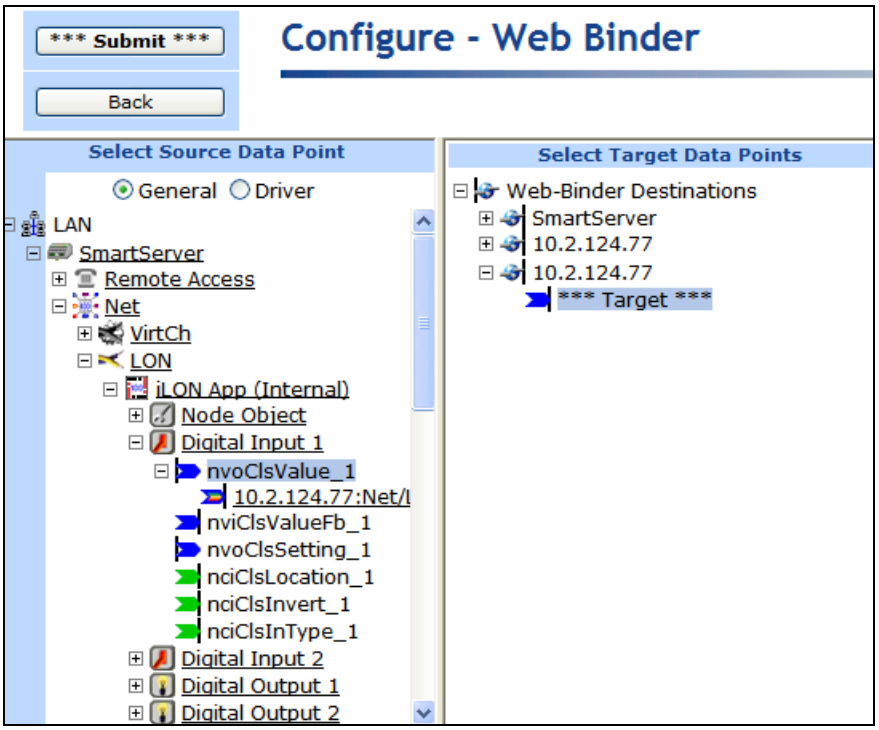
12. Create a Web connection between a source data point on your SmartServer and the WebBinder Target Server. To do this follow these steps:

    a. From the navigation pane in the left frame of the SmartServer Web interface, right-click a source data point and then click **Add Binding** in the shortcut menu.



    b. The **Configure – WebBinder** Web page opens and the hostnames of the local SmartServer and the WebBinder Target server appear in the application frame to the right. The host devices in the right frame are collectively referred to as *WebBinder Destinations*.

    c. From the Webbinder Destinations tree on the right frame, expand the WebBinder Target Server containing the target data points to be connected and then click the ***Target*** item below it.



d. Click **Submit**.

13. Return to your .NET project, put a break point on the first line in *Write* function, and run the project in debug mode. When you change the value of the source data point you selected in step 12, your server-side code's break point should be hit.

14. Attach a text based file attachment such as event log to the Web connection, and run the server-side code in debug mode again. You can see the contents of the file as a text stream in the output window of the debugger as you step through the code.

21 Programming Examples

This chapter includes programming examples, written in Visual C# (.NET 3.5 and .NET 2.0 Frameworks) and Visual Basic with Microsoft Visual Studio 2008, that demonstrate how to use the SmartServer's SOAP API to create custom applications. These programming examples create simple console applications that do the following:

- Read and write data point values.
- Create and read a data logger.
- Create a scheduler and a calendar.
- Create and install LONWORKS devices
- Commission unconfigured external devices.
- Discover and install uncommissioned external devices.
- Configure the SmartServer (with System Service Methods).

Notes:

All examples assume that you are using a SmartServer that has been set to its factory default settings. This prevents compilation errors based on mismatching <UCPTname> properties of the objects in the LONWORKS network hierarchy (*network*/*channel*/*device*/*functional block*/*data point*).

You can download these programming examples from the *i*.LON SmartServer Community Web site at *ilonsmartserver.com*.

21.1 Visual C#.NET Examples

21.1.1 Reading and Writing Data Point Values in Visual C# .NET

This C# console example toggles the SmartServer's digital relay outputs when run. It demonstrates how to use an xSelect statement to filter items returned by a *List*() method, and it demonstrates how to write to data points using values and presets.

You can execute this code after you have referenced and inherited from the SmartServer WSDL as described in section 20.1, and instantiated and initialized the Web service client as described in section 20.2.

For more information on the data point properties set and read in this example, see section 4.3.2, *Using the Get Function on the Data Server*, and *section 4.3.3, Using the Read Function on the Data Server*, respectively.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SmartServerConsoleExample
{
class DpProgram
{
    // If you are using NET 2.0 Framework, uncomment the following line of code to enter your
    // SmartServer's IP Address

    // public static string _iLonEndpointIpAddress = "<SmartServer IP Address>";

    static void Main(string[] args)
     {
            iLON_SoapCalls.BindClientToSmartServer();

            // If you are using NET 2.0 Framework, comment out the previous line of code, and then
            // uncomment the following line of code

            // iLON_SoapCalls.BindClientToSmartServer(_iLonEndpointIpAddress);

            iLON_SmartServer.iLON100portTypeClient SmartServer = iLON_SoapCalls._iLON;
```

```
        try
        {
            // See Section 20.2.1 (NET 3.5) or 20.2.2 (NET 2.0)for more information on iLON_SoapCalls class

            SmartServerConsoleExample.iLON_SmartServer.E_xSelect xSelect =
            new SmartServerConsoleExample.iLON_SmartServer.E_xSelect();

            xSelect.xSelect = "//Item[@xsi:type=\"Dp_Cfg\"][contains(UCPTaliasName,\"nviClaValue\")]";

            iLON_SmartServer.Item_Coll ItemColl = SmartServer.List(xSelect);
            iLON_SmartServer.Item_DataColl ItemDataColl = SmartServer.Read(ItemColl);


        if (ItemColl.UCPTfaultCount > 0)
        {
            Console.Out.WriteLine("you've got errors");
        }

        else
        {
            for (int i = 0; i < ItemColl.Item.Length; i++)
            {
                iLON_SmartServer.Item Dps = ItemColl.Item[i];
                Console.Out.WriteLine(Dps.UCPTname);

                     iLON_SmartServer.Dp_Data DpValues =
                     (iLON_SmartServer.Dp_Data)ItemDataColl.Item[i];

                Console.Out.WriteLine(DpValues.UCPTvalue[0].Value);


                if (DpValues.UCPTvalue[0].Value == "0.0 0")
                {
                    DpValues.UCPTvalue[0].Value = "100.0 1";
                    DpValues.UCPTvalue[1].Value = "ON";

                    Console.Out.WriteLine(DpValues.UCPTvalue[0].Value);
                }
                else if (DpValues.UCPTvalue[0].Value == "100.0 1")
                {
                    DpValues.UCPTvalue[0].Value = "0.0 0";
                    DpValues.UCPTvalue[1].Value = "OFF";

                    Console.Out.WriteLine(DpValues.UCPTvalue[0].Value);
                }

            }

            SmartServer.Write(ItemDataColl);
        }

       Console.ReadLine();
    }

    finally
    {
       iLON_SoapCalls.CloseBindingToSmartServer();
    }

   }
  }
}
```

21.1.2 Creating and Reading a Data Logger in Visual C# .NET

The following C# console example creates a data logger and then reads the data recorded by it. You can execute this code after you have referenced and inherited from the SmartServer WSDL as described in section 20.1, and instantiated and initialized the Web service client as described in section 20.2.

21.1.2.1 Creating a Data Logger

The following C# console example creates a new data logger from an existing uninstantiated (hidden) data logger on the SmartServer, specifies the type, format, and size of the new data logger, and then specifies that the data logger record both of the SmartServer's digital relay outputs every minute (the Net/LON/iLON App/Digital Output 1/nviClaValue_1 and Net/LON/iLON App/Digital Output 2/nviClaValue_2 data points).

For more information on the data logger properties set in this example, see section 5.3.2, *Using the Get Function on a Data Logger*.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace SmartServerConsoleExample
{
    class Program
    {
        // If you are using NET 2.0 Framework, uncomment the following line of code to enter your
        // SmartServer's IP Address

        // public static string _iLonEndpointIpAddress = "<SmartServer IP Address>";

        static void PrintGetError(iLON_SmartServer.Item_CfgColl ItemCfgColl)
        {
            // print out error and exit
            Console.Out.WriteLine("An error occurred:");
            for (int j = 0; j < ItemCfgColl.Item.Length; j++)
            {
                if (ItemCfgColl.Item[j].fault != null)
                {
                   Console.Out.WriteLine("Item: " + ItemCfgColl.Item[j].UCPTname + ", fault code: " +
                   ItemCfgColl.Item[j].fault.faultcode + ", fault string: " +
                   ItemCfgColl.Item[j].fault.faultstring);
                }
            }
        }

        static void PrintGetError(iLON_SmartServer.Item_Coll ItemColl)
        {
            // print out error and exit
            Console.Out.WriteLine("An error occurred:");
            for (int j = 0; j < ItemColl.Item.Length; j++)
            {
                if (ItemColl.Item[j].fault != null)
                {
                   Console.Out.WriteLine("Item: " + ItemColl.Item[j].UCPTname + ", fault code: " +
                   ItemColl.Item[j].fault.faultcode + ", fault string: " +
                   ItemColl.Item[j].fault.faultstring);
                }
            }
        }

        static void Main(string[] args)
        {
           iLON_SoapCalls.BindClientToSmartServer();

           // If you are using NET 2.0 Framework, comment out the previous line of code, and then
           // uncomment the following line of code

           // iLON_SoapCalls.BindClientToSmartServer(_iLonEndpointIpAddress);

           iLON_SmartServer.iLON100portTypeClient SmartServer = iLON_SoapCalls._iLON;

           try
           {
```

```
 // -------------- CREATING A DATA LOGGER --------------

//Create an xSelect object and then specify the filter to be used

iLON_SmartServer.E_xSelect xSelect =
    new iLON_SmartServer.E_xSelect();

xSelect.xSelect =
  "//Item[@xsi:type=\"LON_Fb_Cfg\"][contains(UCPTname,\"Log\")][UCPThidden = \"1\"]";

//Create an ItemColl that stores objects returned by List()function that takes an xSelect object

iLON_SmartServer.Item_Coll ItemColl = SmartServer.List(xSelect);

//Create an ItemCfgColl that stores the objects to be returned by a Get() function
//that takes the ItemColl returned by the List()

ItemColl.xSelect = "//Item[@xsi:type=\"LON_Fb_Cfg\"]";
iLON_SmartServer.Item_CfgColl ItemCfgColl = SmartServer.Get(ItemColl);

  //check that there are obejcts in the ItemCfgColl

if (ItemCfgColl.UCPTfaultCount > 0)
{
    PrintGetError(ItemCfgColl);
}

else
{
    //Create LON_Fb_Cfg item
    ItemCfgColl.Item[0].UCPThidden = 0;
    ItemCfgColl.Item[0].UCPTname = "Net/LON/iLON App/myDataLogger";
    iLON_SmartServer.Item_Coll ItemColl_SetReturn = SmartServer.Set(ItemCfgColl);

    //create new Data Logger from existing one
    iLON_SmartServer.UFPTdataLogger_Cfg myDataLogger =
      new  ConsoleApplication_CSharp_Test_3._5.iLON_SmartServer.UFPTdataLogger_Cfg();
    myDataLogger.UCPTname = "Net/LON/iLON App/myDataLogger";
    myDataLogger.UCPTannotation = "#8000010128000000[4].UFPTdataLogger";
    myDataLogger.UCPTlogFileName = "Net/LON/iLON App/myDataLogger.csv";
    myDataLogger.UCPTlogSize = 100;
    myDataLogger.UCPTlogLevelAlarm = 50;

    myDataLogger.UCPTlogType =
      new ConsoleApplication_CSharp_Test_3._5.iLON_SmartServer.E_LonString();
    myDataLogger.UCPTlogType.Value = "LT_HISTORICAL";
    myDataLogger.UCPTlogType.LonFormat = "UCPTlogType";

    myDataLogger.UCPTlogFormat =
      new ConsoleApplication_CSharp_Test_3._5.iLON_SmartServer.E_LonString();
    myDataLogger.UCPTlogFormat.Value = "LF_TEXT";
    myDataLogger.UCPTlogFormat.LonFormat = "UCPTlogFormat";

    //create DP reference array to store data points by new Data Logger
    myDataLogger.DataPoint =
      new ConsoleApplication_CSharp_Test_3._5.iLON_SmartServer.E_DpRef[2];

    //speficy data points to be logged by new Data Logger

    iLON_SmartServer.UFPTdataLogger_DpRef dataPointRef1 =
      new iLON_SmartServer.UFPTdataLogger_DpRef();
    dataPointRef1.UCPTname = "Net/LON/iLON App/Digital Output 2/nviClaValue_2";
    dataPointRef1.UCPTformatDescription = "#0000000000000000[0].SNVT_switch";
    dataPointRef1.UCPTpollRate = 60;
    dataPointRef1.dpType = "Input";

    iLON_SmartServer.UFPTdataLogger_DpRef dataPointRef2 =
      new iLON_SmartServer.UFPTdataLogger_DpRef();
    dataPointRef2.UCPTname = "Net/LON/iLON App/Digital Output 1/nviClaValue_1";
    dataPointRef2.UCPTformatDescription = "#0000000000000000[0].SNVT_switch";
```

```
                dataPointRef2.UCPTpollRate = 60;
                dataPointRef2.dpType = "Input";

                //store data points in DP reference array
                myDataLogger.DataPoint[0] = dataPointRef1;
                myDataLogger.DataPoint[1] = dataPointRef2;

                //call Set function
                iLON_SmartServer.Item_CfgColl itemCfgColl = new iLON_SmartServer.Item_CfgColl();
                itemCfgColl.Item = new iLON_SmartServer.Item_Cfg[1];
                itemCfgColl.Item[0] = myDataLogger;


                iLON_SmartServer.Item_Coll ItemColl_Set_DataLogger_Return =
                 SmartServer.Set(itemCfgColl);

                if (ItemColl_Set_DataLogger_Return.UCPTfaultCount > 0)
                {
                    PrintGetError(ItemColl_Set_DataLogger_Return);
                }

                else
                {
                    iLON_SmartServer.Item newDataLogger = ItemColl_Set_DataLogger_Return.Item[0];
                    Console.WriteLine("New Data Logger = " + newDataLogger.UCPTname);
                }

            }
               Console.ReadLine();
        }

        finally
        {
           iLON_SoapCalls.CloseBindingToSmartServer();
        }

       }
     }
   }
```

21.1.2.2 Reading a Data Logger

The following C# console example reads and prints out the last 10 entries for one of the two data points recorded by the new data logger you created in the previous section, *Creating a Data Logger*. For more information on the data logger properties used in this example, see section 5.3.4, *Using the Read Function on a Data Logger*.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace SmartServerConsoleExample
{
    class Program
    {
        // If you are using NET 2.0 Framework, uncomment the following line of code to enter your
        // SmartServer's IP Address

        // public static string _iLonEndpointIpAddress = "<SmartServer IP Address>";

         static void PrintGetError(iLON_SmartServer.Item_Coll ItemColl)
         {
             // print out error and exit
             Console.Out.WriteLine("An error occurred:");
             for (int j = 0; j < ItemColl.Item.Length; j++)
             {
                 if (ItemColl.Item[j].fault != null)
                 {
```

```
            Console.Out.WriteLine("Item: " + ItemColl.Item[j].UCPTname + ", fault code: " +
            ItemColl.Item[j].fault.faultcode + ", fault string: " +
            ItemColl.Item[j].fault.faultstring);
        }
    }
}

static void Main(string[] args)
{
    iLON_SoapCalls.BindClientToSmartServer();

   // If you are using NET 2.0 Framework, comment out the previous line of code, and then
   // uncomment the following line of code

   // iLON_SoapCalls.BindClientToSmartServer(_iLonEndpointIpAddress);

    iLON_SmartServer.iLON100portTypeClient SmartServer = iLON_SoapCalls._iLON;

   try
   {

    // -------------- READING A DATA LOGGER --------------

    //Create an xSelect object and then specify the UCPTname of the Data Logger created
   // in the previous section as the filter ("Net/LON/iLON App/myDataLogger")

    ConsoleApplication_CSharp_Test_3._5.iLON_SmartServer.E_xSelect xSelect =
       new ConsoleApplication_CSharp_Test_3._5.iLON_SmartServer.E_xSelect();

    xSelect.xSelect = "//Item[UCPTname = \"Net/LON/iLON App/myDataLogger\"]";

    //Create an ItemColl that stores objects returned by List()function that takes an xSelect object

    iLON_SmartServer.Item_Coll ItemColl = SmartServer.List(xSelect);

   //check that there are obejcts in the ItemColl

    if (ItemColl.UCPTfaultCount > 0)
    {
        PrintGetError(ItemColl);
    }
    else
    {
         iLON_SmartServer.Item myDataLogger = ItemColl.Item[0];
         Console.WriteLine("Data Logger = " + myDataLogger.UCPTname + "\r\n");
    }

    //we use an xSelect to read only the last 10 records in the Data Logger for one data point
    ItemColl.xSelect = "//Item
       [UCPTpointName=\"Net/LON/iLON App/Digital Output 1/nviClaValue_1\"]
       [position()>=last()-10]";

    // Read Data Logger
    iLON_SmartServer.Item_DataColl dataLogger = SmartServer.Read(ItemColl);

     for (int i = 0; i < dataLogger.Item.Length; i++)
        {
            iLON_SmartServer.UFPTdataLogger_Data dataLoggerDataCheck =
              dataLogger.Item[i] as iLON_SmartServer.UFPTdataLogger_Data;

            if (dataLoggerDataCheck != null)
            {
                iLON_SmartServer.UFPTdataLogger_Data dataLoggerData =
                   (iLON_SmartServer.UFPTdataLogger_Data)dataLogger.Item[i];

                Console.Out.WriteLine(dataLoggerData.UCPTname + " was " +
                  dataLoggerData.UCPTvalue[0].Value + " at " +
                  dataLoggerData.UCPTlastUpdate + "\r\n");
            }
        }
```

```
            Console.ReadLine();
            }

            finally
            {
              iLON_SoapCalls.CloseBindingToSmartServer();
            }

          }
      }
}
```

21.1.3 Creating a Scheduler and Calendar in Visual C# .NET

This C# console example creates a Scheduler and Calendar for hypothetically controlling the lighting and heating of a store. You can execute this code after you have referenced and inherited from the SmartServer WSDL as described in section 20.1, and instantiated and initialized the Web service client as described in section 20.2.

The example creates a new Scheduler from an existing uninstantiated (hidden) Scheduler on the SmartServer. It creates separate daily schedules for weekdays, Saturdays, and Sundays, and it specifies that the scheduler turn on and off the SmartServer's digital relay outputs (the Net/LON/iLON App/Digital Output 1/nviClaValue_1 and Net/LON/iLON App/Digital Output 2/nviClaValue_2 data points) at specific times based on the day of the week. The example then creates an exception that keeps the lighting and heating off on holidays.

After creating the Scheduler, this example either gets the Calendar on the SmartServer if it has already been instantiated or creates a new Calendar. The example then specifies the dates of the holidays for the exception created in the Scheduler, and it specifies over how many years the holiday exceptions are to occur.

For more information on the Scheduler and Calendar properties set in this example, see section 9.3.2, *Using the Get Function a Scheduler* and section 10.3.2, *Using the Get Function a Calendar*, respectively.

Note: The <UCPTexceptionName> property is the unique identifier for exceptions defined in the Scheduler and Calendar. This means that the <UCPTexceptionName> property of new exceptions you create must be unique to the Calendar; otherwise, the exception you create will overwrite an existing exception. To prevent overwriting an existing exception, you can loop through the existing exceptions on the Calendar and check whether the <UCPTexceptionName> property of the exception you are creating matches that of any existing exceptions. This example assumes that your SmartServer has been set to its factory default settings and therefore does not perform this check.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace SmartServerConsoleExample
{
    class Program
    {
        // If you are using NET 2.0 Framework, uncomment the following line of code to enter your
        // SmartServer's IP Address

        // public static string _iLonEndpointIpAddress = "<SmartServer IP Address>";

        static void PrintGetError(iLON_SmartServer.Item_CfgColl ItemCfgColl)
        {
            // print out error and exit
            Console.Out.WriteLine("An error occurred:");
            for (int j = 0; j < ItemCfgColl.Item.Length; j++)
            {
                if (ItemCfgColl.Item[j].fault != null)
                {
```

```
            Console.Out.WriteLine("Item: " + ItemCfgColl.Item[j].UCPTname + ", fault code: "
              + ItemCfgColl.Item[j].fault.faultcode + ", fault string: " +
              ItemCfgColl.Item[j].fault.faultstring);
        }
    }
}

 static void PrintGetError(iLON_SmartServer.Item_Coll ItemColl)
{
    // print out error and exit
    Console.Out.WriteLine("An error occurred:");
    for (int j = 0; j < ItemColl.Item.Length; j++)
    {
        if (ItemColl.Item[j].fault != null)
        {
            Console.Out.WriteLine("Item: " + ItemColl.Item[j].UCPTname + ", fault code: "
              + ItemColl.Item[j].fault.faultcode + ", fault string: " +
              ItemColl.Item[j].fault.faultstring);
        }
    }
}

static void Main(string[] args)
{
   iLON_SoapCalls.BindClientToSmartServer();

   // If you are using NET 2.0 Framework, comment out the previous line of code, and then
   // uncomment the following line of code

   // iLON_SoapCalls.BindClientToSmartServer(_iLonEndpointIpAddress);

   iLON_SmartServer.iLON100portTypeClient SmartServer = iLON_SoapCalls._iLON;

   try
   {

    // -------------- CREATING A SCHEDULER --------------

   //Create an xSelect object and then specify the filter to be used

   iLON_SmartServer.E_xSelect xSelect = new iLON_SmartServer.E_xSelect();
   xSelect.xSelect = "//Item[@xsi:type=\"LON_Fb_Cfg\"]
                      [contains(UCPTname,\"Scheduler\")][UCPThidden = \"1\"]";

   //Create an ItemColl that stores objects returned by List()function that takes an xSelect object

   iLON_SmartServer.Item_Coll ItemColl = SmartServer.List(xSelect);

   //Create an ItemCfgColl that stores the objects to be returned by a Get() function that
   //takes the ItemColl returned by the List()

   ItemColl.xSelect = "//Item[@xsi:type=\"LON_Fb_Cfg\"]";
   iLON_SmartServer.Item_CfgColl ItemCfgColl = SmartServer.Get(ItemColl);

     //check that there are obejcts in the ItemCfgColl

   if (ItemCfgColl.UCPTfaultCount > 0)
   {
       PrintGetError(ItemCfgColl);
   }

   else
   {
      //Create LON_Fb_Cfg item
       ItemCfgColl.Item[0].UCPThidden = 0;
       ItemCfgColl.Item[0].UCPTname = "Net/LON/iLON App/myScheduler";
       iLON_SmartServer.Item_Coll ItemColl_SetReturn = SmartServer.Set(ItemCfgColl);


       //create new Scheduler from existing one
       iLON_SmartServer.UFPTscheduler_Cfg myScheduler = new iLON_SmartServer.UFPTscheduler_Cfg();
```

```
myScheduler.UCPTname = "Net/LON/iLON App/myScheduler";
myScheduler.UCPTannotation = "#8000010128000000[4].UFPTscheduler";

//create DP reference array to store data points controlled by new Scheduler
myScheduler.DataPoint = new iLON_SmartServer.E_DpRef[2];

//speficy data points to be controlled by new Scheduler

iLON_SmartServer.UFPTscheduler_DpRef dataPointRef1 = new iLON_SmartServer.UFPTscheduler_DpRef();
dataPointRef1.UCPTname = "Net/LON/iLON App/Digital Output 1/nviClaValue_1";
dataPointRef1.UCPTformatDescription = "#0000000000000000[0].SNVT_switch";
dataPointRef1.SCPTdelayTime = 0;
dataPointRef1.SCPTdelayTimeSpecified = true;
dataPointRef1.dpType = "Output";


iLON_SmartServer.UFPTscheduler_DpRef dataPointRef2 = new iLON_SmartServer.UFPTscheduler_DpRef();
dataPointRef2.UCPTname = "Net/LON/iLON App/Digital Output 2/nviClaValue_2";
dataPointRef2.UCPTformatDescription = "#0000000000000000[0].SNVT_switch";
dataPointRef2.SCPTdelayTime = 0;
dataPointRef2.SCPTdelayTimeSpecified = true;
dataPointRef2.dpType = "Output";

//store data points in DP reference array
myScheduler.DataPoint[0] = dataPointRef1;
myScheduler.DataPoint[1] = dataPointRef2;

//set range of dates in which Scheduler is effective
iLON_SmartServer.UFPTscheduler_CfgEffectivePeriod effectivePeriod =
    new iLON_SmartServer.UFPTscheduler_CfgEffectivePeriod();
effectivePeriod.StartDate = new DateTime(2009, 6, 8);
effectivePeriod.EndDate = new DateTime(2020, 12, 31);
effectivePeriod.StartDateSpecified = true;
effectivePeriod.EndDateSpecified = true;
myScheduler.ScheduleEffectivePeriod = effectivePeriod;

//create daily schedule for weekdays
iLON_SmartServer.UFPTscheduler_CfgDayBased dayBasedSchedule_weekdays =
    new iLON_SmartServer.UFPTscheduler_CfgDayBased();
dayBasedSchedule_weekdays.UCPTindex = 0;
dayBasedSchedule_weekdays.UCPTindexSpecified = true;
dayBasedSchedule_weekdays.UCPTdescription = "Weekday";
dayBasedSchedule_weekdays.UCPTpriority = 255;

//create events for weekday schedule

dayBasedSchedule_weekdays.Event = new iLON_SmartServer.UFPTscheduler_CfgEvent[2];
dayBasedSchedule_weekdays.Event[0] = new iLON_SmartServer.UFPTscheduler_CfgEvent();
dayBasedSchedule_weekdays.Event[1] = new iLON_SmartServer.UFPTscheduler_CfgEvent();

//---create ON event----
iLON_SmartServer.UFPTscheduler_CfgEvent onEvent = new iLON_SmartServer.UFPTscheduler_CfgEvent();
onEvent.UCPTindex = 0;
onEvent.UCPTindexSpecified = true;
onEvent.UCPTtime = new DateTime(2009, 6, 8, 10, 00, 00);

onEvent.UCPTvalue = new iLON_SmartServer.E_LonString[1];
onEvent.UCPTvalue[0] = new iLON_SmartServer.E_LonString();
onEvent.UCPTvalue[0].Value = "ON";
onEvent.UCPTvalue[0].LonFormat = "UCPTvalueDef";

dayBasedSchedule_weekdays.Event[0] = onEvent;

//---create OFF event---
iLON_SmartServer.UFPTscheduler_CfgEvent offEvent = new iLON_SmartServer.UFPTscheduler_CfgEvent();
offEvent.UCPTindex = 1;
offEvent.UCPTindexSpecified = true;
offEvent.UCPTtime = new DateTime(2009, 6, 8, 21, 00, 00);

offEvent.UCPTvalue = new iLON_SmartServer.E_LonString[1];
offEvent.UCPTvalue[0] = new iLON_SmartServer.E_LonString();
```

```
offEvent.UCPTvalue[0].Value = "OFF";
offEvent.UCPTvalue[0].LonFormat = "UCPTvalueDef";

dayBasedSchedule_weekdays.Event[1] = offEvent;

//set Monday--Friday as the days in this daily schedule
iLON_SmartServer.UFPTscheduler_CfgDayBasedWeekdays mon_to_fri =
    new iLON_SmartServer.UFPTscheduler_CfgDayBasedWeekdays();
mon_to_fri.UCPTmonday = 1;
mon_to_fri.UCPTtuesday = 1;
mon_to_fri.UCPTwednesday = 1;
mon_to_fri.UCPTthursday = 1;
mon_to_fri.UCPTfriday = 1;

mon_to_fri.UCPTsaturday = 0;
mon_to_fri.UCPTsunday = 0;

dayBasedSchedule_weekdays.Weekdays = mon_to_fri;

//create daily schedule for Saturdays

iLON_SmartServer.UFPTscheduler_CfgDayBased dayBasedSchedule_Sat =
    new iLON_SmartServer.UFPTscheduler_CfgDayBased();
dayBasedSchedule_Sat.UCPTindex = 1;
dayBasedSchedule_Sat.UCPTindexSpecified = true;
dayBasedSchedule_Sat.UCPTdescription = "Saturday";
dayBasedSchedule_Sat.UCPTpriority = 255;

//create events for Saturday schedule

dayBasedSchedule_Sat.Event = new iLON_SmartServer.UFPTscheduler_CfgEvent[2];
dayBasedSchedule_Sat.Event[0] = new iLON_SmartServer.UFPTscheduler_CfgEvent();
dayBasedSchedule_Sat.Event[1] = new iLON_SmartServer.UFPTscheduler_CfgEvent();

//---create ON event----
iLON_SmartServer.UFPTscheduler_CfgEvent onEvent_Sat =
    new iLON_SmartServer.UFPTscheduler_CfgEvent();
onEvent_Sat.UCPTindex = 0;
onEvent_Sat.UCPTindexSpecified = true;
onEvent_Sat.UCPTtime = new DateTime(2009, 6, 8, 10, 00, 00);

onEvent_Sat.UCPTvalue = new iLON_SmartServer.E_LonString[1];
onEvent_Sat.UCPTvalue[0] = new iLON_SmartServer.E_LonString();
onEvent_Sat.UCPTvalue[0].Value = "ON";
onEvent_Sat.UCPTvalue[0].LonFormat = "UCPTvalueDef";

dayBasedSchedule_Sat.Event[0] = onEvent_Sat;

//---create OFF event---
iLON_SmartServer.UFPTscheduler_CfgEvent offEvent_Sat =
    new iLON_SmartServer.UFPTscheduler_CfgEvent();
offEvent_Sat.UCPTindex = 1;
offEvent_Sat.UCPTindexSpecified = true;
offEvent_Sat.UCPTtime = new DateTime(2009, 6, 8, 19, 00, 00);

offEvent_Sat.UCPTvalue = new iLON_SmartServer.E_LonString[1];
offEvent_Sat.UCPTvalue[0] = new iLON_SmartServer.E_LonString();
offEvent_Sat.UCPTvalue[0].Value = "OFF";
offEvent_Sat.UCPTvalue[0].LonFormat = "UCPTvalueDef";

dayBasedSchedule_Sat.Event[1] = offEvent_Sat;

//set Saturday as only day in this daily schedule
iLON_SmartServer.UFPTscheduler_CfgDayBasedWeekdays sat =
    new iLON_SmartServer.UFPTscheduler_CfgDayBasedWeekdays();

sat.UCPTsaturday = 1;

sat.UCPTsunday = 0;
sat.UCPTmonday = 0;
sat.UCPTtuesday = 0;
```

```
sat.UCPTwednesday = 0;
sat.UCPTthursday = 0;
sat.UCPTfriday = 0;

dayBasedSchedule_Sat.Weekdays = sat;

//create daily schedule for Sundays
iLON_SmartServer.UFPTscheduler_CfgDayBased dayBasedSchedule_Sun =
    new iLON_SmartServer.UFPTscheduler_CfgDayBased();
dayBasedSchedule_Sun.UCPTindex = 2;
dayBasedSchedule_Sun.UCPTindexSpecified = true;
dayBasedSchedule_Sun.UCPTdescription = "Sunday";
dayBasedSchedule_Sun.UCPTpriority = 255;

//create events for Sunday Schedule

dayBasedSchedule_Sun.Event = new iLON_SmartServer.UFPTscheduler_CfgEvent[2];
dayBasedSchedule_Sun.Event[0] = new iLON_SmartServer.UFPTscheduler_CfgEvent();
dayBasedSchedule_Sun.Event[1] = new iLON_SmartServer.UFPTscheduler_CfgEvent();

//---create ON event----
iLON_SmartServer.UFPTscheduler_CfgEvent onEvent_Sun =
    new iLON_SmartServer.UFPTscheduler_CfgEvent();
onEvent_Sun.UCPTindex = 0;
onEvent_Sun.UCPTindexSpecified = true;
onEvent_Sun.UCPTtime = new DateTime(2009, 6, 8, 12, 00, 00);

onEvent_Sun.UCPTvalue = new iLON_SmartServer.E_LonString[1];
onEvent_Sun.UCPTvalue[0] = new iLON_SmartServer.E_LonString();
onEvent_Sun.UCPTvalue[0].Value = "ON";
onEvent_Sun.UCPTvalue[0].LonFormat = "UCPTvalueDef";

dayBasedSchedule_Sun.Event[0] = onEvent_Sun;

//---create OFF event---
iLON_SmartServer.UFPTscheduler_CfgEvent offEvent_Sun =
    new iLON_SmartServer.UFPTscheduler_CfgEvent();
offEvent_Sun.UCPTindex = 1;
offEvent_Sun.UCPTindexSpecified = true;
offEvent_Sun.UCPTtime = new DateTime(2009, 6, 8, 18, 00, 00);

offEvent_Sun.UCPTvalue = new iLON_SmartServer.E_LonString[1];
offEvent_Sun.UCPTvalue[0] = new iLON_SmartServer.E_LonString();
offEvent_Sun.UCPTvalue[0].Value = "OFF";
offEvent_Sun.UCPTvalue[0].LonFormat = "UCPTvalueDef";

dayBasedSchedule_Sun.Event[1] = offEvent_Sun;

//set Sunday as only day in this daily schedule
iLON_SmartServer.UFPTscheduler_CfgDayBasedWeekdays sun =
    new iLON_SmartServer.UFPTscheduler_CfgDayBasedWeekdays();
sun.UCPTsunday = 1;

sun.UCPTsaturday = 0;
sun.UCPTmonday = 0;
sun.UCPTtuesday = 0;
sun.UCPTwednesday = 0;
sun.UCPTthursday = 0;
sun.UCPTfriday = 0;

dayBasedSchedule_Sun.Weekdays = sun;

//store daily schedules we created in a DayBased[]

myScheduler.DayBased = new iLON_SmartServer.UFPTscheduler_CfgDayBased[3];

myScheduler.DayBased[0] = new iLON_SmartServer.UFPTscheduler_CfgDayBased();
myScheduler.DayBased[0] = dayBasedSchedule_weekdays;

myScheduler.DayBased[1] = new iLON_SmartServer.UFPTscheduler_CfgDayBased();
myScheduler.DayBased[1] = dayBasedSchedule_Sat;
```

```
myScheduler.DayBased[2] = new iLON_SmartServer.UFPTscheduler_CfgDayBased();
myScheduler.DayBased[2] = dayBasedSchedule_Sun;

//create a date-based schedule (an exception) for some American Holidays

//*** NOTE: You must use Calendar application to specify the dates in which this
//alterntate schedule is applicable***

iLON_SmartServer.UFPTscheduler_CfgDateBased holidays =
    new iLON_SmartServer.UFPTscheduler_CfgDateBased();
holidays.UCPTindex = 0;
holidays.UCPTindexSpecified = true;
holidays.UCPTpriority = 250;

//create events for Holiday schedule

holidays.Event = new iLON_SmartServer.UFPTscheduler_CfgEvent[3];
holidays.Event[0] = new iLON_SmartServer.UFPTscheduler_CfgEvent();
holidays.Event[1] = new iLON_SmartServer.UFPTscheduler_CfgEvent();
holidays.Event[2] = new iLON_SmartServer.UFPTscheduler_CfgEvent();

//create LOCK event at 00:00 for Holiday schedule
iLON_SmartServer.UFPTscheduler_CfgEvent lockEvent_holiday =
    new iLON_SmartServer.UFPTscheduler_CfgEvent();
lockEvent_holiday.UCPTindex = 0;
lockEvent_holiday.UCPTindexSpecified = true;
lockEvent_holiday.UCPTtime = new DateTime(2009, 6, 8, 00, 00, 00);

lockEvent_holiday.UCPTeventType = new iLON_SmartServer.E_LonString();
lockEvent_holiday.UCPTeventType.Value = "ET_LOCK";
lockEvent_holiday.UCPTeventType.LonFormat = "UCPTeventType";

holidays.Event[0] = lockEvent_holiday;

//create ON event for Holiday schedule
iLON_SmartServer.UFPTscheduler_CfgEvent onEvent_holiday =
    new iLON_SmartServer.UFPTscheduler_CfgEvent();
onEvent_holiday.UCPTindex = 1;
onEvent_holiday.UCPTindexSpecified = true;
onEvent_holiday.UCPTtime = new DateTime(2009, 6, 8, 12, 00, 00);

onEvent_holiday.UCPTeventType = new iLON_SmartServer.E_LonString();
onEvent_holiday.UCPTeventType.Value = "ET_NUL";
onEvent_holiday.UCPTeventType.LonFormat = "UCPTeventType";

onEvent_holiday.UCPTvalue = new iLON_SmartServer.E_LonString[1];
onEvent_holiday.UCPTvalue[0] = new iLON_SmartServer.E_LonString();
onEvent_holiday.UCPTvalue[0].Value = "ON";
onEvent_holiday.UCPTvalue[0].LonFormat = "UCPTvalueDef";

holidays.Event[1] = onEvent_holiday;

//create OFF event for Holiday schedule

iLON_SmartServer.UFPTscheduler_CfgEvent offEvent_holiday =
    new iLON_SmartServer.UFPTscheduler_CfgEvent();
offEvent_holiday.UCPTindex = 2;
offEvent_holiday.UCPTindexSpecified = true;
offEvent_holiday.UCPTtime = new DateTime(2009, 6, 8, 18, 00, 00);

offEvent_holiday.UCPTeventType = new iLON_SmartServer.E_LonString();
offEvent_holiday.UCPTeventType.Value = "ET_NUL";
offEvent_holiday.UCPTeventType.LonFormat = "UCPTeventType";

offEvent_holiday.UCPTvalue = new iLON_SmartServer.E_LonString[1];
offEvent_holiday.UCPTvalue[0] = new iLON_SmartServer.E_LonString();
offEvent_holiday.UCPTvalue[0].Value = "OFF";
offEvent_holiday.UCPTvalue[0].LonFormat = "UCPTvalueDef";

holidays.Event[2] = offEvent_holiday;
```

```
            //create Exception item
            holidays.Exception = new iLON_SmartServer.UFPTscheduler_CfgDateBasedException[1];
            holidays.Exception[0] = new iLON_SmartServer.UFPTscheduler_CfgDateBasedException();
            holidays.Exception[0].UCPTexceptionName = "Holidays";
            holidays.Exception[0].UCPTindex = 0;
            holidays.Exception[0].UCPTindexSpecified = true;

            //store date-based (exception) schedule we created in a DateBased[]

            myScheduler.DateBased = new iLON_SmartServer.UFPTscheduler_CfgDateBased[1];

            myScheduler.DateBased[0] = new iLON_SmartServer.UFPTscheduler_CfgDateBased();
            myScheduler.DateBased[0] = holidays;

            //call Set function
            iLON_SmartServer.Item_CfgColl itemCfgColl = new iLON_SmartServer.Item_CfgColl();
            itemCfgColl.Item = new iLON_SmartServer.Item_Cfg[1];
            itemCfgColl.Item[0] = myScheduler;


            iLON_SmartServer.Item_Coll ItemColl_Set_Scheduler_Return = SmartServer.Set(itemCfgColl);

            if (ItemColl_Set_Scheduler_Return.UCPTfaultCount > 0)
            {
                PrintGetError(ItemColl);
            }

            else
            {
                iLON_SmartServer.Item newScheduler = ItemColl_Set_Scheduler_Return.Item[0];
                Console.WriteLine("New Scheduler = " + newScheduler.UCPTname);
            }

        }

    // -------------- CREATING A CALENDAR --------------

        //Create a new UFPTcalendar_Cfg item
        myCalendar = new iLON_SmartServer.UFPTcalendar_Cfg();
        myCalendar.UCPTname = "Net/LON/iLON App/myCalendar";
        myCalendar.UCPTannotation = "#8000010128000000[4].UFPTcalendar";

        //Configure the Calendar
        iLON_SmartServer.UFPTscheduler_CfgEffectivePeriod effectivePeriod_calendar =
            new iLON_SmartServer.UFPTscheduler_CfgEffectivePeriod();
        effectivePeriod_calendar.StartDate = new DateTime(2009, 6, 8);
        effectivePeriod_calendar.EndDate = new DateTime(2020, 12, 31);
        effectivePeriod_calendar.StartDateSpecified = true;
        effectivePeriod_calendar.EndDateSpecified = true;
        myCalendar.ScheduleEffectivePeriod = effectivePeriod_calendar;

        //create an exception
        myCalendar.Exception = new iLON_SmartServer.UFPTcalendar_CfgException[1];
        myCalendar.Exception[0] = new iLON_SmartServer.UFPTcalendar_CfgException();

        myCalendar.Exception[0].UCPTexceptionName = "Holidays";
        myCalendar.Exception[0].UCPTaliasName = "Holidays";
        myCalendar.Exception[0].UCPTindex = 0;
        myCalendar.Exception[0].UCPTindexSpecified = true;
        myCalendar.Exception[0].UCPTtemporary = 0;
        myCalendar.Exception[0].UCPTtemporarySpecified = true;
        myCalendar.Exception[0].UCPTmaxClient = 1;
        myCalendar.Exception[0].UCPTmaxClientSpecified = true;

        myCalendar.Exception[0].Client = new iLON_SmartServer.UFPTcalendar_CfgExceptionClient[1];
        myCalendar.Exception[0].Client[0] = new iLON_SmartServer.UFPTcalendar_CfgExceptionClient();
        myCalendar.Exception[0].Client[0].UCPTname = "Net/LON/iLON App/myScheduler";
        myCalendar.Exception[0].Client[0].UCPTservicePath =
          new ConsoleApplication_CSharp_Test_3._5.iLON_SmartServer.E_Path();
        myCalendar.Exception[0].Client[0].UCPTservicePath.Value = "";
```

```
//create exception dates
myCalendar.Exception[0].Schedule = new iLON_SmartServer.UFPTcalendar_CfgExceptionSchedule[4];
myCalendar.Exception[0].Schedule[0] = new iLON_SmartServer.UFPTcalendar_CfgExceptionSchedule();
myCalendar.Exception[0].Schedule[1] = new iLON_SmartServer.UFPTcalendar_CfgExceptionSchedule();
myCalendar.Exception[0].Schedule[2] = new iLON_SmartServer.UFPTcalendar_CfgExceptionSchedule();
myCalendar.Exception[0].Schedule[3] = new iLON_SmartServer.UFPTcalendar_CfgExceptionSchedule();

//create 4th of July exception
//==========================

myCalendar.Exception[0].Schedule[0].UCPTschedMonth = new iLON_SmartServer.E_LonString();
myCalendar.Exception[0].Schedule[0].UCPTschedMonth.LonFormat = "UCPTschedMonth";
myCalendar.Exception[0].Schedule[0].UCPTschedMonth.Value = "MN_JUL";

myCalendar.Exception[0].Schedule[0].UCPTschedDay = new iLON_SmartServer.E_LonString();
myCalendar.Exception[0].Schedule[0].UCPTschedDay.LonFormat = "UCPTschedDay";
myCalendar.Exception[0].Schedule[0].UCPTschedDay.Value = "DM_DAY_4";

//set start date
myCalendar.Exception[0].Schedule[0].StartDate = new iLON_SmartServer.UFPTcalendar_CfgESDate();
myCalendar.Exception[0].Schedule[0].StartDate.UCPTdate = new DateTime(2009, 6, 8);

//set end date
myCalendar.Exception[0].Schedule[0].EndDate = new iLON_SmartServer.UFPTcalendar_CfgESDate();
myCalendar.Exception[0].Schedule[0].EndDate.UCPTdate = new DateTime(2020, 12, 31);

//create Labor Day exception
//==========================

myCalendar.Exception[0].Schedule[1].UCPTschedMonth = new iLON_SmartServer.E_LonString();
myCalendar.Exception[0].Schedule[1].UCPTschedMonth.LonFormat = "UCPTschedMonth";
myCalendar.Exception[0].Schedule[1].UCPTschedMonth.Value = "MN_SEP";

myCalendar.Exception[0].Schedule[1].UCPTschedDay = new iLON_SmartServer.E_LonString();
myCalendar.Exception[0].Schedule[1].UCPTschedDay.LonFormat = "UCPTschedDay";
myCalendar.Exception[0].Schedule[1].UCPTschedDay.Value = "DM_FIRST_MON";

//set start date
myCalendar.Exception[0].Schedule[1].StartDate = new iLON_SmartServer.UFPTcalendar_CfgESDate();
myCalendar.Exception[0].Schedule[1].StartDate.UCPTdate = new DateTime(2009, 6, 8);

//set end date
myCalendar.Exception[0].Schedule[1].EndDate = new iLON_SmartServer.UFPTcalendar_CfgESDate();
myCalendar.Exception[0].Schedule[1].EndDate.UCPTdate = new DateTime(2020, 12, 31);

//create Thanksgiving exception
//==========================

myCalendar.Exception[0].Schedule[2].UCPTschedMonth = new iLON_SmartServer.E_LonString();
myCalendar.Exception[0].Schedule[2].UCPTschedMonth.LonFormat = "UCPTschedMonth";
myCalendar.Exception[0].Schedule[2].UCPTschedMonth.Value = "MN_NOV";

myCalendar.Exception[0].Schedule[2].UCPTschedDay = new iLON_SmartServer.E_LonString();
myCalendar.Exception[0].Schedule[2].UCPTschedDay.LonFormat = "UCPTschedDay";
myCalendar.Exception[0].Schedule[2].UCPTschedDay.Value = "DM_FOURTH_THU";

//set start date
myCalendar.Exception[0].Schedule[2].StartDate = new iLON_SmartServer.UFPTcalendar_CfgESDate();
myCalendar.Exception[0].Schedule[2].StartDate.UCPTdate = new DateTime(2009, 6, 8);

//set end date
myCalendar.Exception[0].Schedule[2].EndDate = new iLON_SmartServer.UFPTcalendar_CfgESDate();
myCalendar.Exception[0].Schedule[2].EndDate.UCPTdate = new DateTime(2020, 12, 31);

//Create Christmas exception
//==========================

myCalendar.Exception[0].Schedule[3].UCPTschedMonth = new iLON_SmartServer.E_LonString();
myCalendar.Exception[0].Schedule[3].UCPTschedMonth.LonFormat = "UCPTschedMonth";
myCalendar.Exception[0].Schedule[3].UCPTschedMonth.Value = "MN_DEC";
```

```
            myCalendar.Exception[0].Schedule[3].UCPTschedDay = new iLON_SmartServer.E_LonString();
            myCalendar.Exception[0].Schedule[3].UCPTschedDay.LonFormat = "UCPTschedDay";
            myCalendar.Exception[0].Schedule[3].UCPTschedDay.Value = "DM_DAY_25";

            //set start date
            myCalendar.Exception[0].Schedule[3].StartDate = new iLON_SmartServer.UFPTcalendar_CfgESDate();
            myCalendar.Exception[0].Schedule[3].StartDate.UCPTdate = new DateTime(2009, 6, 8);

            //set end date
            myCalendar.Exception[0].Schedule[3].EndDate = new iLON_SmartServer.UFPTcalendar_CfgESDate();
            myCalendar.Exception[0].Schedule[3].EndDate.UCPTdate = new DateTime(2020, 12, 31);

            //call Set function
            iLON_SmartServer.Item_CfgColl itemCfgColl_Calendar = new iLON_SmartServer.Item_CfgColl();
            itemCfgColl_Calendar.Item = new iLON_SmartServer.Item_Cfg[1];
            itemCfgColl_Calendar.Item[0] = myCalendar;


            iLON_SmartServer.Item_Coll ItemColl_Set_Calendar_Return =
                SmartServer.Set(itemCfgColl_Calendar);

            if (ItemColl_Set_Calendar_Return.UCPTfaultCount > 0)
            {
                PrintGetError(ItemColl);
            }

            else
            {
                iLON_SmartServer.Item newCalendar = ItemColl_Set_Calendar_Return.Item[0];
                Console.WriteLine("Calendar used for this Scheduler is " + newCalendar.UCPTname);
            }

        Console.ReadLine();
        }

        finally
        {
           iLON_SoapCalls.CloseBindingToSmartServer();
        }

       }
    }
 }
```

21.1.4 Creating and Installing a LONWORKS Device in Visual C# .NET

This C# console example creates two LONWORKS devices, and then it commissions the devices, starts the devices' applications, and gets the devices' templates (to display the devices' functional blocks and data points in the SmartServer Web interface). The example then prints out the names and statuses of the devices that have been installed. Note that you need to replace the values of the <UCPTname>, <UCPTuniqueID>, <UCPTprogramID>, and <UCPTurlTemplate> properties provided in this example with those of the devices you are creating and installing.

You can execute this code after you have referenced and inherited from the SmartServer WSDL as described in section 20.1, and instantiated and initialized the Web service client as described in section 20.2. You must also upload the device interface (XIF) files of the devices you are creating to the root/LonWorks/import folder on the SmartServer flash disk.

For more information on the LONWORKS device properties set in this example, see section 14.3.2, *Using the Get Function on a LonWorks Device*. For more information on the network management commands issues in this example, see section 14.3.3.1, *Issuing Network Management Commands*.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication_LON_Device
{
```

```
class Program
{
     //Function required for converting device Neuron IDs and program IDs to a byte[]
    static public byte[] HexStringToArray(string str)
    {
        int nLen = str.Length / 2;
        byte[] arr = new byte[nLen];

        for (int i = 0; i < nLen; i++)
        {
            string strByte = str.Substring(i * 2, 2);
            arr[i] = Byte.Parse(strByte, System.Globalization.NumberStyles.HexNumber);
        }
        return arr;
    }
    static void Main(string[] args)
    {
        iLON_SoapCalls.BindClientToSmartServer();
        iLON_SmartServer.iLON100portTypeClient SmartServer = iLON_SoapCalls._iLON;

       try
       {

        // -------------- CREATING LONWORKS DEVICES --------------

        //Create a new LON_Device_Cfg Item

        iLON_SmartServer.LON_Device_Cfg my_LON_Device1 = new iLON_SmartServer.LON_Device_Cfg();
        iLON_SmartServer.LON_Device_Cfg my_LON_Device2 = new iLON_SmartServer.LON_Device_Cfg();

        //Create an ItemCfgColl to store the LON Devices we just created

        iLON_SmartServer.Item_CfgColl ItemCfgColl = new iLON_SmartServer.Item_CfgColl();
        ItemCfgColl.Item = new iLON_SmartServer.Item_Cfg[2];
        ItemCfgColl.Item[0] = my_LON_Device1;
        ItemCfgColl.Item[1] = my_LON_Device2;

        //=====CREATING AND INSTALLING LON DEVICE #1==================

        // specify properties of new LON Device #1
        my_LON_Device1.UCPTname = "Net/LON/DIO-1";
        my_LON_Device1.UCPTlocal = 0;
        my_LON_Device1.UCPTuniqueId = HexStringToArray("00a145791500");
        my_LON_Device1.UCPTprogramId = HexStringToArray("80000105288a0403");
        my_LON_Device1.UCPTurlTemplate = "/root/lonWorks/Import/Echelon/LonPoint/Version3/dio-10v3.XIF";

        my_LON_Device1.UCPTcommissionStatus = new iLON_SmartServer.E_LonString();
        my_LON_Device1.UCPTcommissionStatus.Value = "COMMISSIONED";
        my_LON_Device1.UCPTapplicationStatus = new iLON_SmartServer.E_LonString();
        my_LON_Device1.UCPTapplicationStatus.Value = "APP_RUNNING";

        //create a command array to store device commands to be sent
        my_LON_Device1.Command = new iLON_SmartServer.LON_Device_CfgCommand[3];

        //commission device
        my_LON_Device1.Command[0] =
           new ConsoleApplication_LON_Device.iLON_SmartServer.LON_Device_CfgCommand();
        my_LON_Device1.Command[0].UCPTcommand = new iLON_SmartServer.LON_Device_eCommand();
        my_LON_Device1.Command[0].UCPTcommand =  iLON_SmartServer.LON_Device_eCommand.ChangeCommissionStatus;
        my_LON_Device1.Command[0].UCPTstatus =
           new ConsoleApplication_LON_Device.iLON_SmartServer.E_LonString();
        my_LON_Device1.Command[0].UCPTstatus.LonFormat = "UCPTstatus";
        my_LON_Device1.Command[0].UCPTstatus.Value = "STATUS_REQUEST";

        //run device application
        my_LON_Device1.Command[1] = new iLON_SmartServer.LON_Device_CfgCommand();
        my_LON_Device1.Command[1].UCPTcommand = new iLON_SmartServer.LON_Device_eCommand();
        my_LON_Device1.Command[1].UCPTcommand = iLON_SmartServer.LON_Device_eCommand.ChangeApplicationStatus;
        my_LON_Device1.Command[1].UCPTstatus =
           new ConsoleApplication_LON_Device.iLON_SmartServer.E_LonString();
        my_LON_Device1.Command[1].UCPTstatus.LonFormat = "UCPTstatus";
```

```
my_LON_Device1.Command[1].UCPTstatus.Value = "STATUS_REQUEST";

//get the device template to show FBs and DPs in Web UI
my_LON_Device1.Command[2] = new iLON_SmartServer.LON_Device_CfgCommand();
my_LON_Device1.Command[2].UCPTcommand = new iLON_SmartServer.LON_Device_eCommand();
my_LON_Device1.Command[2].UCPTcommand = iLON_SmartServer.LON_Device_eCommand.GetTemplate;
my_LON_Device1.Command[2].UCPTstatus =
   new ConsoleApplication_LON_Device.iLON_SmartServer.E_LonString();
my_LON_Device1.Command[2].UCPTstatus.LonFormat = "UCPTstatus";
my_LON_Device1.Command[2].UCPTstatus.Value = "STATUS_REQUEST";

//=====CREATING AND INSTALLING LON DEVICE #2==================

// specify properties of new LON Device #2
my_LON_Device2.UCPTname = "Net/LON/DIO-2";
my_LON_Device2.UCPTlocal = 0;
my_LON_Device2.UCPTuniqueId = HexStringToArray("00a145784600");
my_LON_Device2.UCPTprogramId = HexStringToArray("80000105288a0403");
my_LON_Device2.UCPTurlTemplate = "/root/lonWorks/Import/Echelon/LonPoint/Version3/dio-10v3.XIF";
my_LON_Device2.UCPTcommissionStatus = new iLON_SmartServer.E_LonString();
my_LON_Device2.UCPTcommissionStatus.Value = "COMMISSIONED";
my_LON_Device2.UCPTapplicationStatus = new iLON_SmartServer.E_LonString();
my_LON_Device2.UCPTapplicationStatus.Value = "APP_RUNNING";

//create a command array to store device commands to be sent
my_LON_Device2.Command = new iLON_SmartServer.LON_Device_CfgCommand[3];

//commission device
my_LON_Device2.Command[0] = new iLON_SmartServer.LON_Device_CfgCommand();
my_LON_Device2.Command[0].UCPTcommand = new iLON_SmartServer.LON_Device_eCommand();
my_LON_Device2.Command[0].UCPTcommand = iLON_SmartServer.LON_Device_eCommand.ChangeCommissionStatus;
my_LON_Device2.Command[0].UCPTstatus =
   new ConsoleApplication_LON_Device.iLON_SmartServer.E_LonString();
my_LON_Device2.Command[0].UCPTstatus.LonFormat = "UCPTstatus";
my_LON_Device2.Command[0].UCPTstatus.Value = "STATUS_REQUEST";

//run device application
my_LON_Device2.Command[1] = new iLON_SmartServer.LON_Device_CfgCommand();
my_LON_Device2.Command[1].UCPTcommand = new iLON_SmartServer.LON_Device_eCommand();
my_LON_Device2.Command[1].UCPTcommand = iLON_SmartServer.LON_Device_eCommand.ChangeApplicationStatus;
my_LON_Device2.Command[1].UCPTstatus =
   new ConsoleApplication_LON_Device.iLON_SmartServer.E_LonString();
my_LON_Device2.Command[1].UCPTstatus.LonFormat = "UCPTstatus";
my_LON_Device2.Command[1].UCPTstatus.Value = "STATUS_REQUEST";

//get the device template to show FBs and DPs in Web UI
my_LON_Device2.Command[2] = new iLON_SmartServer.LON_Device_CfgCommand();
my_LON_Device2.Command[2].UCPTcommand = new iLON_SmartServer.LON_Device_eCommand();
my_LON_Device2.Command[2].UCPTcommand = iLON_SmartServer.LON_Device_eCommand.GetTemplate;
my_LON_Device2.Command[2].UCPTstatus =
   new ConsoleApplication_LON_Device.iLON_SmartServer.E_LonString();
my_LON_Device2.Command[2].UCPTstatus.LonFormat = "UCPTstatus";
my_LON_Device2.Command[2].UCPTstatus.Value = "STATUS_REQUEST";

//Call the Set() function

iLON_SmartServer.Item_Coll Device_Return_ItemColl = SmartServer.Set(ItemCfgColl);

Device_Return_ItemColl.xSelect = "//Item[@xsi:type=\"LON_Device_Cfg\"]";

if (Device_Return_ItemColl.UCPTfaultCount > 0)
{
    // print out error and exit
    Console.Out.WriteLine("An error occurred:");

    for (int j = 0; j < Device_Return_ItemColl.Item.Length; j++)
    {
        if (Device_Return_ItemColl.Item[j].fault != null)
        {
            Console.Out.WriteLine("Item: " + Device_Return_ItemColl.Item[j].UCPTname +
              ", fault code: " + Device_Return_ItemColl.Item[j].fault.faultcode +
```

```
                        ", fault string: " + Device_Return_ItemColl.Item[j].fault.faultstring);
                }
            }
        }
        else
        {
            ItemCfgColl = SmartServer.Get(Device_Return_ItemColl);

            for (int j = 0; j < ItemCfgColl.Item.Length; j++)
            {
                iLON_SmartServer.LON_Device_Cfg newDevice = (iLON_SmartServer.LON_Device_Cfg)ItemCfgColl.Item[j];
                Console.WriteLine("New Device Created = " + newDevice.UCPTname + ".  Status = " +
                  newDevice.UCPTcommissionStatus.Value + " and " +
                  newDevice.UCPTapplicationStatus.Value + ".\r");
            }
        }
     Console.ReadLine();
     }

     finally
     {
       iLON_SoapCalls.CloseBindingToSmartServer();
     }

    }
  }
}
```

21.1.5 Commissioning External Devices in Visual C# .NET

This C# console example reads the <UCPTitemStatus> of external LonWorks devices (obtained using an xSelect), commissions any unconfigured devices, and reports the status of the network management commands.

You can execute this code after you have referenced and inherited from the SmartServer WSDL as described in section 20.1, and instantiated and initialized the Web service client as described in section 20.2.

For more information on the network management commands issued in this example, see section 14.3.3.1, *Issuing Network Management Commands*.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace SmartServerConsoleExample
{
    class MyDeviceProgram
    {
        // If you are using NET 2.0 Framework, uncomment the following line of code to enter your
        // SmartServer's IP Address

        // public static string _iLonEndpointIpAddress = "<SmartServer IP Address>";

        static void PrintGetError(iLON_SmartServer.Item_CfgColl ItemCfgColl)
        {
            // print out error and exit
            Console.Out.WriteLine("An error occurred:");
            for (int j = 0; j < ItemCfgColl.Item.Length; j++)
            {
                if (ItemCfgColl.Item[j].fault != null)
                {
                   Console.Out.WriteLine("Item: " + ItemCfgColl.Item[j].UCPTname + ", fault code: " +
                   ItemCfgColl.Item[j].fault.faultcode + ", fault string: " +
                   ItemCfgColl.Item[j].fault.faultstring);
                }
            }
        }
```

```
        // If you are using NET 2.0 Framework, uncomment the following line of code to enter your
        // SmartServer's IP Address

        // public static string _iLonEndpointIpAddress = "<SmartServer IP Address>";


    static void Main(string[] args)
    {
        iLON_SoapCalls.BindClientToSmartServer();

        // If you are using NET 2.0 Framework, comment out the previous line of code, and then
        // uncomment the following line of code

        // iLON_SoapCalls.BindClientToSmartServer(_iLonEndpointIpAddress);

        iLON_SmartServer.iLON100portTypeClient SmartServer = iLON_SoapCalls._iLON;

        // See Section 20.2.1 (NET 3.5) or 20.2.2 (NET 2.0)for more information on iLON_SoapCalls class

        try
        {

         //------------Commissioning Uncommissioned External LonWorks Devices---------------------

         Console.Out.WriteLine("Commissioning Uncommissioned External LonWorks Devices with an
         xSelect\r\n ------------------------------\r\n ------------------------------\r\n");


         //we create an xSelect object and then specify the filter to be used

        iLON_SmartServer.E_xSelect xSelect =
          new SmartServerConsoleExample.iLON_SmartServer.E_xSelect();

        xSelect.xSelect =
        "//Item[@xsi:type=\"LON_Device_Cfg\"][UCPTitemStatus=\"IS_UNCONFIGURED\"]";

         //Create an ItemColl that stores objects returned by List()function that takes an xSelect
        //object

         iLON_SmartServer.Item_Coll ItemColl = SmartServer.List(xSelect);

         //we use an xSelect to further filter the items returned by the List() function
         ItemColl.xSelect = "//Item[@xsi:type=\"LON_Device_Cfg\"][UCPTlocal =\"0\"]";

         //we create an ItemCfgColl that stores the objects returned by a Get() function that
        //takes the ItemColl returned by the List()

         iLON_SmartServer.Item_CfgColl ItemCfgColl = SmartServer.Get(ItemColl);

         //check that there are obejcts in the ItemCfgColl

         if (ItemCfgColl.UCPTfaultCount > 0)
         {
             PrintGetError(ItemCfgColl);
         }

         else
         {

             for (int i = 0; i < ItemCfgColl.Item.Length; i++)
             {
                 iLON_SmartServer.LON_Device_Cfg deviceItems =
                (iLON_SmartServer.LON_Device_Cfg)ItemCfgColl.Item[i];

                 Console.Out.WriteLine(deviceItems.UCPTname + ", STATUS = " +
                 deviceItems.UCPTitemStatus.Value + "\r\n");

                 deviceItems.UCPTcommissionStatus.Value = "COMMISSIONED";
                 deviceItems.UCPTapplicationStatus.Value = "APP_RUNNING";
```

```
            deviceItems.Command[0].UCPTcommand =
            iLON_SmartServer.LON_Device_eCommand.ChangeApplicationStatus;
            deviceItems.Command[0].UCPTstatus.Value = "STATUS_REQUEST";

            deviceItems.Command[1].UCPTcommand =
            iLON_SmartServer.LON_Device_eCommand.ChangeCommissionStatus;
            deviceItems.Command[1].UCPTstatus.Value = "STATUS_REQUEST";

            deviceItems.Command[2].UCPTcommand = iLON_SmartServer.LON_Device_eCommand.Reset;
            deviceItems.Command[2].UCPTstatus.Value = "STATUS_REQUEST";

            Console.Out.WriteLine("*DEVICE CONFIGURATION CHECK**\r\n\r\n" +
            deviceItems.UCPTname + "\r\n---------------------------------\r\n\r\n" + "STATUS
            = " + deviceItems.UCPTitemStatus.Value + "\r\n COMMISSION STATUS = " +
            deviceItems.UCPTcommissionStatus.Value + "\r\n APPLICATION STATUS = " +
            deviceItems.UCPTapplicationStatus.Value + "\r\n");

        }
    }

    iLON_SmartServer.Item_Coll ItemColl_SetReturn = SmartServer.Set(ItemCfgColl);

    ItemColl_SetReturn.xSelect = "//Item[@xsi:type=\"LON_Device_Cfg\"]";

    bool bAllDone = false;
    do
    {
        ItemCfgColl = SmartServer.Get(ItemColl_SetReturn);

        if (ItemCfgColl.UCPTfaultCount > 0)
        {
            PrintGetError(ItemCfgColl);
            break;
        }
        bAllDone = true;

        // now check all the items to make sure all the commands are done
        for (int i = 0; i < ItemCfgColl.Item.Length; i++)
        {
            iLON_SmartServer.LON_Device_Cfg deviceItemsCheck =
           (iLON_SmartServer.LON_Device_Cfg)ItemCfgColl.Item[i];

           bool bOnlinePass = (deviceItemsCheck.Command[0].UCPTstatus.Value == "STATUS_DONE");
           bool bCommissionPass = (deviceItemsCheck.Command[1].UCPTstatus.Value == "STATUS_DONE");
           bool bResetPass = (deviceItemsCheck.Command[2].UCPTstatus.Value == "STATUS_DONE");
           bool bOnlineFail = (deviceItemsCheck.Command[0].UCPTstatus.Value == "STATUS_DONE");
           bool bCommissionFail = (deviceItemsCheck.Command[1].UCPTstatus.Value == "STATUS_DONE");
           bool bResetFail = (deviceItemsCheck.Command[2].UCPTstatus.Value == "STATUS_DONE");

            // print out status
            Console.Out.WriteLine("\r\n *INSTALLATION STATUS CHECK**\r\n\r\n");

            Console.Out.WriteLine(deviceItemsCheck.UCPTname + "ONLINE REQUEST STATUS = " +
            deviceItemsCheck.Command[0].UCPTstatus.Value);

            if (bOnlineFail && deviceItemsCheck.Command[0].fault != null)
            {
                  Console.Out.WriteLine("Error string: " +
                  deviceItemsCheck.Command[0].fault.faultstring + ", Error Code" +
                  deviceItemsCheck.Command[0].fault.faultcode);
            }

                  Console.Out.WriteLine(deviceItemsCheck.UCPTname + "COMMISSION REQUEST
                  STATUS = " + deviceItemsCheck.Command[1].UCPTstatus.Value);
                  if (bOnlineFail && deviceItemsCheck.Command[1].fault != null)
            {
                  Console.Out.WriteLine("Error string: " +
                  deviceItemsCheck.Command[1].fault.faultstring + ", Error Code" +
                  deviceItemsCheck.Command[1].fault.faultcode);
            }
```

```
                    Console.Out.WriteLine(deviceItemsCheck.UCPTname + "RESET REQUEST STATUS =
                    " + deviceItemsCheck.Command[2].UCPTstatus.Value);
                    if (bOnlineFail && deviceItemsCheck.Command[2].fault != null)
                {
                    Console.Out.WriteLine("Error string: " +
                    deviceItemsCheck.Command[2].fault.faultstring + ", Error Code" +
                    deviceItemsCheck.Command[2].fault.faultcode);
                }

                if ((bOnlinePass || bOnlineFail) &&
                    (bCommissionPass || bCommissionFail) &&
                    (bResetPass || bResetFail))
                {
                    // this device is done

                }
                else
                {
                    bAllDone = false;
                    break;
                }
            }

            if (!bAllDone)
            {
                Thread.Sleep(15000);
            }
        }
        while (!bAllDone);
    }
    finally
    {
      iLON_SoapCalls.CloseBindingToSmartServer();
    }

  }
 }
}
```

21.1.6 Discovering and Installing External Devices in Visual C# .NET

This console example scans a LONWORKS network for uncommissioned devices, processes the Neuron ID and program ID data of the discovered devices, and then commissions the devices, starts the devices' applications, and gets the devices' templates (to display the devices' functional blocks and data points in the SmartServer Web interface). The example then prints out the names and statuses of the devices that have been installed.

You can execute this code after you have referenced and inherited from the SmartServer WSDL as described in section 20.1, and instantiated and initialized the Web service client as described in section 20.2. You must also upload the device interface (XIF) files of the devices you are discovering and installing to the root/LonWorks/import folder on the SmartServer flash disk, or create device templates (XML files) for the devices.

For more information on discovering uncommissioned LONWORKS devices, see section 14.1.3.2, *Issuing Network Scan Commands to Discover Devices*.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace ConsoleApplication_CSharp_Test_3._5
{
    class Program
    {

        static public byte[] HexStringToArray(string str)
```

```
{
    int nLen = str.Length / 2;
    byte[] arr = new byte[nLen];

    for (int i = 0; i < nLen; i++)
    {
        string strByte = str.Substring(i * 2, 2);
        arr[i] = Byte.Parse(strByte, System.Globalization.NumberStyles.HexNumber);
    }
    return arr;
}

static void Main(string[] args)
{
    iLON_SoapCalls.BindClientToSmartServer();
    iLON_SmartServer.iLON100portTypeClient SmartServer = iLON_SoapCalls._iLON;

    try
    {
        //create LONNetworkScanCommandInvoke item and ScanCommand attribute
        iLON_SmartServer.LON_Network_ScanCommand_Invoke networkScan =
           new iLON_SmartServer.LON_Network_ScanCommand_Invoke();
        networkScan.ScanCommand = iLON_SmartServer.LON_Network_eScanCommand.SetScan;

        //***set LONNetworkScanCommandInvoke properties***

        //1. Set network UCPTname
        networkScan.UCPTname = "Net";

        //2. Set Scan Command

        //a. set scan frequency
        iLON_SmartServer.LON_Network_ScanCommand_InvokeCommand scanFrequency =
           new iLON_SmartServer.LON_Network_ScanCommand_InvokeCommand();
        scanFrequency.UCPTcommand = iLON_SmartServer.LON_Device_IlonNi_eCommand.ScanOnce;

        //b. set scan status
        iLON_SmartServer.E_LonString scanStatus = new iLON_SmartServer.E_LonString();
        scanStatus.LonFormat = "UCPTstatus";
        scanStatus.Value = "STATUS_REQUEST";
        scanFrequency.UCPTstatus = scanStatus;

        //c. add scan command to LONNetworkScanCommandInvoke item
        networkScan.Command = new iLON_SmartServer.LON_Network_ScanCommand_InvokeCommand[1];
        networkScan.Command[0] = scanFrequency;

        //3. Set UCPTscan
        iLON_SmartServer.E_LonString domain = new iLON_SmartServer.E_LonString();
        domain.LonFormat = "ucptScan";
        domain.Value = "NST_ILON_DOMAIN";
        networkScan.UCPTscan = new iLON_SmartServer.E_LonString[1];
        networkScan.UCPTscan[0] = domain;

        //send InvokeCmd

        iLON_SmartServer.Item_Coll itemColl = new iLON_SmartServer.Item_Coll();
        itemColl.Item = new iLON_SmartServer.Item[1];
        itemColl.Item[0] = networkScan;
        SmartServer.InvokeCmd(ref itemColl);

        Console.WriteLine("starting scan");

        //send the GetScan command to check network scan progress
        iLON_SmartServer.LON_Network_ScanCommand_Invoke networkScan_Check =
           new iLON_SmartServer.LON_Network_ScanCommand_Invoke();
        networkScan_Check.ScanCommand = iLON_SmartServer.LON_Network_eScanCommand.GetScan;
        networkScan_Check.UCPTname = "Net";

        iLON_SmartServer.Item_Coll itemColl_Check = new iLON_SmartServer.Item_Coll();
        itemColl_Check.Item = new iLON_SmartServer.Item[1];
        itemColl_Check.Item[0] = networkScan_Check;
```

```
//Check scan status
bool scanDone = false;
while (!scanDone)
{
    SmartServer.InvokeCmd(ref itemColl_Check);

    iLON_SmartServer.InvokeCmdResponse scanCheck_Response =
          new iLON_SmartServer.InvokeCmdResponse();
    scanCheck_Response.iLonItem = itemColl_Check;
    iLON_SmartServer.LON_Network_ScanCommand_Invoke scanStatusCheck =
     (iLON_SmartServer.LON_Network_ScanCommand_Invoke)scanCheck_Response.iLonItem.Item[0];

    //if the scan is done set scanDone flag to true
    if (scanStatusCheck.Command[0].UCPTstatus.Value == "STATUS_DONE")
    {
        Console.WriteLine("Network Scan Status = " +
          scanStatusCheck.Command[0].UCPTstatus.Value);
        scanDone = true;
    }

    //if the scan is not done, keep scanDone flag at false, wait 10 seconds, and check again
    else if (scanStatusCheck.Command[0].UCPTstatus.Value == "STATUS_PENDING")
    {
        Console.WriteLine("Network Scan Status = " +
           scanStatusCheck.Command[0].UCPTstatus.Value);
        Thread.Sleep(10000);
    }
}
// A "<network>/#DeviceDiscovery" data logger is automatically created by the network scan
// read the Data Logger and process the data of the discovered data
iLON_SmartServer.UFPTdataLogger_Data deviceDiscovered =
   new iLON_SmartServer.UFPTdataLogger_Data();
deviceDiscovered.UCPTname = "Net/#DeviceDiscovery";
iLON_SmartServer.Item_Coll itemColl_DataLog = new iLON_SmartServer.Item_Coll();
itemColl_DataLog.xSelect = "//Item[@xsi:type=\"UFPTdataLogger_Data\"]";
itemColl_DataLog.Item = new iLON_SmartServer.Item[1];
itemColl_DataLog.Item[0] = deviceDiscovered;

iLON_SmartServer.Item_DataColl dataLogger = SmartServer.Read(itemColl_DataLog);
Console.WriteLine("Devices Discovered = " + (dataLogger.Item.Length - 1));
Console.WriteLine("============================================================");

iLON_SmartServer.Item_CfgColl itemCfgColl = new iLON_SmartServer.Item_CfgColl();

//Create a new ItemCfgColl to store discovered devices
itemCfgColl.Item = new iLON_SmartServer.Item_Cfg[dataLogger.Item.Length - 1];

for (int i = 1; i < dataLogger.Item.Length; i++)
//we start at 1 to account for the metaData item in Data Logger
{
    iLON_SmartServer.UFPTdataLogger_Data dataLoggerData =
      (iLON_SmartServer.UFPTdataLogger_Data)dataLogger.Item[i];

    if (dataLoggerData != null)
    {
        Console.WriteLine("Device #" + i + ": Neuron ID and Program ID = " +
          dataLoggerData.UCPTvalue[0].Value);
    }
    // -------------- CREATING DISCOVERED LONWORKS DEVICES--------------

    //Create a new LON_Device_Cfg Item and add it to ItemCfgColl
    iLON_SmartServer.LON_Device_Cfg my_LON_Device =
          new iLON_SmartServer.LON_Device_Cfg();

     itemCfgColl.Item[i - 1] = my_LON_Device;
    //subtract 1 for the metaData item in Data Logger

    //parse Neuron ID and Program ID from Data Logger
    String NID_PID = dataLoggerData.UCPTvalue[0].Value;
    String NID = NID_PID.Substring(0, 12);
```

```
Console.WriteLine("Neuron ID = " + NID);
String PID = NID_PID.Substring(13, 16);
Console.WriteLine("Program ID = " + PID);

//set Neuron ID, which is a byte[]
my_LON_Device.UCPTuniqueId = (HexStringToArray(NID));

//set Program ID, which is a byte[]
my_LON_Device.UCPTprogramId = (HexStringToArray(PID));

//set template
iLON_SmartServer.E_xSelect xSelect = new iLON_SmartServer.E_xSelect();
xSelect.xSelect = "//Item[@xsi:type=\"TemplateManager_Cfg\"]
                   [UCPTfileType=\"TEMPLATE_OR_XIF\"]
                   [UCPTprogramId=\"" + PID + "\"]";
itemColl = SmartServer.List(xSelect);

iLON_SmartServer.TemplateManager_Surrogate_Cfg template =
 (iLON_SmartServer.TemplateManager_Surrogate_Cfg)itemColl.Item[0];
String templateName = template.UCPTname;
Console.WriteLine("Device Template = " + templateName);
my_LON_Device.UCPTurlTemplate = templateName;

//set the device name

//1. get the name of the channel ("Net/LON")
xSelect.xSelect = "//Item[@xsi:type=\"LON_Channel_Cfg\"][UCPThidden=0]";
itemColl = SmartServer.List(xSelect);
iLON_SmartServer.Item channel = itemColl.Item[0];

//2. get the name of the xif
string[] separator = new string[] { "/" };
String[] templateName_justxif = templateName.Split(separator, 0);
int templateNameLength = templateName_justxif.Length;
String xifName = templateName_justxif[templateNameLength - 1];
Console.WriteLine("XIF Name = " + xifName);

//3. name device using channel name, /device [index], and xif name
// ("Net/LON/Device 1 (ai-10v3.xif)")
String deviceName = channel.UCPTname + "/" + "Device " + i + " (" + xifName + ")";
Console.WriteLine("Device Name = " + deviceName);
Console.WriteLine("============================================================");
my_LON_Device.UCPTname = deviceName;

//set Commission status
iLON_SmartServer.E_LonString commissionStatus_LonString =
  new iLON_SmartServer.E_LonString();
commissionStatus_LonString.Value = "COMMISSIONED";
my_LON_Device.UCPTcommissionStatus = commissionStatus_LonString;

//set Application status
iLON_SmartServer.E_LonString applicationStatus_LonString =
  new iLON_SmartServer.E_LonString();
applicationStatus_LonString.Value = "APP_RUNNING";
my_LON_Device.UCPTapplicationStatus = applicationStatus_LonString;

//set tree and app icon; based on program ID
my_LON_Device.UCPTannotation = PID;


//++++send device commands++++++

//commission device
//create a command array to store device commands to be sent
my_LON_Device.Command = new iLON_SmartServer.LON_Device_CfgCommand[3];

//commission device
my_LON_Device.Command[0] = new iLON_SmartServer.LON_Device_CfgCommand();
my_LON_Device.Command[0].UCPTcommand =
  new iLON_SmartServer.LON_Device_eCommand();
my_LON_Device.Command[0].UCPTcommand =
```

```
            iLON_SmartServer.LON_Device_eCommand.ChangeCommissionStatus;
        my_LON_Device.Command[0].UCPTstatus = new iLON_SmartServer.E_LonString();
        my_LON_Device.Command[0].UCPTstatus.LonFormat = "UCPTstatus";
        my_LON_Device.Command[0].UCPTstatus.Value = "STATUS_REQUEST";

        //run device application
        my_LON_Device.Command[1] = new iLON_SmartServer.LON_Device_CfgCommand();
        my_LON_Device.Command[1].UCPTcommand = new iLON_SmartServer.LON_Device_eCommand();
        my_LON_Device.Command[1].UCPTcommand =
          iLON_SmartServer.LON_Device_eCommand.ChangeApplicationStatus;
        my_LON_Device.Command[1].UCPTstatus = new iLON_SmartServer.E_LonString();
        my_LON_Device.Command[1].UCPTstatus.LonFormat = "UCPTstatus";
        my_LON_Device.Command[1].UCPTstatus.Value = "STATUS_REQUEST";

        //get the device template to show FBs and DPs in Web UI
        my_LON_Device.Command[2] = new iLON_SmartServer.LON_Device_CfgCommand();
        my_LON_Device.Command[2].UCPTcommand = new iLON_SmartServer.LON_Device_eCommand();
        my_LON_Device.Command[2].UCPTcommand =
          iLON_SmartServer.LON_Device_eCommand.GetTemplate;
        my_LON_Device.Command[2].UCPTstatus = new iLON_SmartServer.E_LonString();
        my_LON_Device.Command[2].UCPTstatus.LonFormat = "UCPTstatus";
        my_LON_Device.Command[2].UCPTstatus.Value = "STATUS_REQUEST";
    }

    //Call the Set() function

    iLON_SmartServer.Item_Coll Device_Return_ItemColl = SmartServer.Set(itemCfgColl);

    Device_Return_ItemColl.xSelect = "//Item[@xsi:type=\"LON_Device_Cfg\"]";

    if (Device_Return_ItemColl.UCPTfaultCount > 0)
    {
        // print out error and exit
        Console.WriteLine("An error occurred:");

        for (int j = 0; j < Device_Return_ItemColl.Item.Length; j++)
        {
            if (Device_Return_ItemColl.Item[j].fault != null)
            {
                Console.WriteLine("Item: " + Device_Return_ItemColl.Item[j].UCPTname +
                      ", fault code: " + Device_Return_ItemColl.Item[j].fault.faultcode +
                      ", fault string: " +
                      Device_Return_ItemColl.Item[j].fault.faultstring);
            }
        }
    }
    else
    {
        itemCfgColl = SmartServer.Get(Device_Return_ItemColl);

        for (int j = 0; j < itemCfgColl.Item.Length; j++)
        {
            if (itemCfgColl.Item[j].fault != null)
            {
                Console.WriteLine("Item: " + itemCfgColl.Item[j].UCPTname
                      + ", fault code: " + itemCfgColl.Item[j].fault.faultcode +
                      ", fault string: " +
                      itemCfgColl.Item[j].fault.faultstring);
            }
            else
            {
                iLON_SmartServer.LON_Device_Cfg newDevice =
                  (iLON_SmartServer.LON_Device_Cfg)itemCfgColl.Item[j];
                Console.WriteLine("New Device Created = " + newDevice.UCPTname +
                      ".  Status = " + newDevice.UCPTcommissionStatus.Value +
                      " and " + newDevice.UCPTapplicationStatus.Value + ".");
            }
        }
    }
    Console.ReadLine();
}
```

```
            finally
            {
                iLON_SoapCalls.CloseBindingToSmartServer();
            }
        }
    }
}
```

21.1.7 Configuring the SmartServer in Visual C# .NET

This console example uses the system information methods in the SmartServer's system WSDL (**iLON100_System.wsdl**) to check the SmartServer's current time and system information and then sets a new time. Note that the **iLON_SoapCalls** class references the **iLON100_System** Web service instead of the **iLON100** Web service. The instantiation of the **iLON100_System** Web service for the NET 3.5 and NET 2.0 Frameworks are presented after this example.

For more information on the system information properties set in this example, see section 19.1, *System Service Methods*.

You can execute this code after you have referenced and inherited from the SmartServer WSDL as described in section 20.1.

Main Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace SystemServiceExample
{
    class System_Time
    {
     // If you are using NET 2.0 Framework, uncomment the following line of code to enter your
     // SmartServer's IP Address

     // public static string _iLonEndpointIpAddress = "<SmartServer IP Address>";

    static void Main(string[] args)
     {
            iLON_SoapCalls.BindClientToSmartServer();

            // If you are using NET 2.0 Framework, comment out the previous line of code, and then
            // uncomment the following line of code

            // iLON_SoapCalls.BindClientToSmartServer(_iLonEndpointIpAddress);

            iLON_SmartServer.iLON100portTypeClient SmartServer = iLON_SoapCalls._iLON;

            // See Section 20.2.1 (NET 3.5) or 20.2.2 (NET 2.0)for more information on iLON_SoapCalls class

            try
            {

             //------------Checking System Time-----------------------------------------
             //This code checks the SmartServer's time and system info and then sets a new time

             Console.Out.WriteLine("Checking the SmartServer's System Time\r\n");

             iLON_SmartServer_System.messageProperties_system time =
              new iLON_SmartServer_System.messageProperties_system();

             string timeData =
            "<iLONSystemService><UCPTsystemInfoType>SI_TIME</UCPTsystemInfoType></iLONSystemService>";
             string timeResult = SmartServer.SystemService_Read_Info(ref time, timeData);
             Console.Out.WriteLine(timeResult);

             Console.Out.WriteLine("\r\nChecking the SmartServer's System Information\r\n");
```

```
            iLON_SmartServer_System.messageProperties_system systemInfo =
            new iLON_SmartServer_System.messageProperties_system();

             string staticData =
            "<iLONSystemService><UCPTsystemInfoType>SI_STATIC</UCPTsystemInfoType></iLONSystemService>";
             string staticResult = SmartServer.SystemService_Read_Info(ref systemInfo, staticData);
             Console.Out.WriteLine(staticResult);

             Console.Out.WriteLine("\r\n Changing the SmartServer's System Time\r\n");

             iLON_SmartServer_System.messageProperties_system revisedTime =
             new iLON_SmartServer_System.messageProperties_system();

             string revisedTimeData =
             "<iLONSystemService><TIME>SI_TIME<UCPTsystemTime>2008-07-05T10:20:00</UCPTsystemTime></TI
             ME></iLONSystemService>";

             string revisedTimeResult = SmartServer.SystemService_Write_Info(ref revisedTime,
             revisedTimeData);

             Console.Out.WriteLine(revisedTimeResult);

             Console.Out.WriteLine("\r\nTake a 10-second break to see if time updates properly \r\n");
             Thread.Sleep(10000);

             iLON_SmartServer_System.messageProperties_system newTime = new
             iLON_SmartServer_System.messageProperties_system();

             string newTimeData =
            "<iLONSystemService><UCPTsystemInfoType>SI_TIME</UCPTsystemInfoType></iLONSystemService>";

             string newTimeResult = SmartServer.SystemService_Read_Info(ref newTime, newTimeData);

            Console.Out.WriteLine(newTimeResult);

            Console.In.ReadLine();
         }

        finally
        {
          iLON_SoapCalls.CloseBindingToSmartServer();
        }

       }
    }
}
```

Web Service Instantiation in iLON_SoapCalls Class for NET 3.5 Framework

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel;

namespace SystemServiceExample
{
    class iLON_SoapCalls
    {
            // your SmartServers's IpAddress
        public static string _iLonEndpointIpAddress = "your SmartServer's IP address";

        // your SmartServer's web service reference
        static public iLON_SmartServer_System.iLON100portTypeClient _iLON = null;


        /// <summary>
        ///   Instantiates the i.LON web service for
        ///   .NET 3.5
        /// </summary>
        static public void BindClientToSmartServer()
```

```
        {
            // Specify the binding to be used for the client.
            BasicHttpBinding binding = new BasicHttpBinding();

            // Initialize the namespace
            binding.Namespace = "http://wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/";

            // Obtain the URL of the Web service on the SmartServer.
            System.ServiceModel.EndpointAddress endpointAddress
              = new System.ServiceModel.EndpointAddress("http://"
              + _iLonEndpointIpAddress +  "/WSDL/iLON100_System.wsdl");

            // Instantiate the i.LON web service object with this address and binding.
            _iLON = new iLON_SmartServer_System.iLON100portTypeClient(binding, endpointAddress);

            // Uncommment the lines below to enable authentication
           //  binding.Security.Mode =
           //  System.ServiceModel.BasicHttpSecurityMode.TransportCredentialOnly;
           //  binding.Security.Transport.ClientCredentialType =
           //  System.ServiceModel.HttpClientCredentialType.Basic;
           //  _iLON.ChannelFactory.Credentials.UserName.UserName = "ilon";
           //  _iLON.ChannelFactory.Credentials.UserName.Password = "ilon";
        }


        /// <summary>
        ///   Close the i.LON web service
        /// </summary>
        static public void CloseBindingToSmartServer()
        {
            // Closing the client gracefully
            // closes the connection and cleans up resources
            try
            {
                _iLON.Close();
            }
            finally
            {
                _iLON = null;
            }
        }
    }
}
```

Web Service Instantiation in iLON_SoapCalls Class for NET 2.0 Framework

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel;

namespace SystemServiceExample
{
    class iLON_SoapCalls
    {

        // your SmartServer's Web service reference
        static public iLON_SmartServer_System _iLON = null;

        /// <summary>
        ///   Instantiates the SmartServer Web service for .NET 2.0
        /// </summary>
        static public void BindClientToSmartServer(string iLonEndpointIpAddress)
        {
            _iLON = new iLON_WebService();
            String strOrigUrl = _iLON.Url;
            _iLON.Url = strOrigUrl.Replace("localhost", iLonEndpointIpAddress);
            _iLON.messagePropertiesValue = new iLON_SmartServer.messageProperties();

            // uncomment the 2 lines below to enable authentication
```

```
            //            _iLON.Credentials = new System.Net.NetworkCredential("ilon", "ilon");
            //            _iLON.PreAuthenticate = true;
        }

    }
}
```

21.2 Visual Basic.NET Examples

21.2.1 Reading and Writing Data Point Values in Visual Basic.NET

This VB console example toggles the SmartServer's digital relay outputs when run. It demonstrates how to use an xSelect statement to filter items returned by a *List*() method, and it demonstrates how to write to data points using values and presets.

You can execute this code after you have referenced and inherited from the SmartServer WSDL as described in section 20.1, and instantiated and initialized the Web service client as described in section 20.2.3.

For more information on the data point properties set and read in this example, see section 4.3.2, *Using the Get Function on the Data Server*, and *section 4.3.3, Using the Read Function on the Data Server*, respectively.

```
Module DpModule

    Sub Main()

        'See Section 20.2.3 for more information on iLON_SoapCalls class
         Dim SmartServer As iLON_SoapCalls = New iLON_SoapCalls
         SmartServer.BindClientToSmartServer()

       Try

        Console.Out.WriteLine(vbNewLine + vbNewLine + "Reading and Writing DP Values with an xSelect"
        + vbNewLine + "-------------------------------" + vbNewLine)

         'we create an xSelect object and then specify the filter to be used
          Dim xSelect As iLON_SmartServer.E_xSelect = New iLON_SmartServer.E_xSelect
         xSelect.xSelect = "//Item[@xsi:type=""Dp_Cfg""][contains (UCPTname,""nviClaValue"")]"

         'Create an ItemColl that stores the objects returned by a List() function that takes our
          'xSelect object
         Dim ItemColl As iLON_SmartServer.Item_Coll = SmartServer._iLON.List(xSelect)

         'we create an ItemDataColl that stores the objects returned by a Read() function that takes
        'the ItemColl returned by the List()
         Dim ItemDataColl As iLON_SmartServer.Item_DataColl = SmartServer._iLON.Read(ItemColl)

         'check that there are obejcts in the ItemDataColl
         If (ItemDataColl.UCPTfaultCount > 0) Then

             Console.Out.WriteLine("you've got Read errors")

         Else

             Console.Out.WriteLine("Reading Data Point Values" + vbNewLine)
             For i As Integer = 0 To ItemDataColl.Item.Length - 1

                 ' we allocate a Item-Data array object to read DP names and values

                 Dim dpItems As iLON_SmartServer.Dp_Data = ItemDataColl.Item(i)
                 Console.Out.WriteLine(dpItems.UCPTname + " = " + dpItems.UCPTvalue(0).Value)

                 If (dpItems.UCPTvalue(0).Value = "100.0 1") Then
                     dpItems.UCPTvalue(0).Value = "0.0 0"
                     dpItems.UCPTvalue(1).Value = "OFF"

                 ElseIf (dpItems.UCPTvalue(0).Value = "0.0 0") Then
                     dpItems.UCPTvalue(0).Value = "100.0 1"
                     dpItems.UCPTvalue(1).Value = "ON"

                 End If
```

```
            Next

            Dim ItemWriteDpValues As ILON_SmartServer.Item_Coll = SmartServer._iLON.Write(ItemDataColl)

            'check that there are obejcts in the ItemWriteDpValues

            If (ItemWriteDpValues.UCPTfaultCount > 0) Then

                Console.Out.WriteLine("you've got errors")

            Else

                Console.Out.WriteLine(vbNewLine + "Reading Updated Data Point Values" + vbNewLine)
                For j As Integer = 0 To ItemWriteDpValues.Item.Length - 1

                 ' we allocate a Item-Data array object to read DP values, which should all be 100.0 1

                    Dim dpItems As iLON_SmartServer.Dp_Data = ItemWriteDpValues.Item(j)
                    Console.Out.WriteLine(dpItems.UCPTname + " = " + dpItems.UCPTvalue(0).Value)

                Next

            End If

        End If

        Console.ReadLine()

      Finally
       SmartServer.CloseBindingToSmartServer()

      End Try
    End Sub
End Module
```

21.2.2 Creating and Reading a Data Logger in Visual Basic. NET

The following VB console example creates a data logger and then reads the data recorded by it. You can execute this code after you have referenced and inherited from the SmartServer WSDL as described in section 20.1, and instantiated and initialized the Web service client as described in section 20.2.

21.2.2.1 Creating a Data Logger

This VB console example creates a new data logger from an existing uninstantiated (hidden) data logger on the SmartServer, specifies the type, format, and size of the new data logger, and then specifies that the data logger record both of the SmartServer's digital relay outputs every minute (the Net/LON/iLON App/Digital Output 1/nviClaValue_1 and Net/LON/iLON App/Digital Output 2/nviClaValue_2 data points).

For more information on the data logger properties set in this example, see section 5.3.2, *Using the Get Function on a Data Logger*.

```
Module DataLogModule

    Private Sub PrintGetError(ByVal ItemCfgColl As iLON_SmartServer.Item_CfgColl)
        ' print out error and exit
        Console.Out.WriteLine("An error occurred:")
        For j As Integer = 0 To ItemCfgColl.Item.Length - 1
            If ItemCfgColl.Item(j).fault IsNot Nothing Then
                Console.Out.WriteLine((("Item: " & ItemCfgColl.Item(j).UCPTname & ", fault code: ") +
                   ItemCfgColl.Item(j).fault.faultcode.Value & ", fault string: ") +
                   ItemCfgColl.Item(j).fault.faultstring)
            End If
        Next
    End Sub

    Public Sub Main()
```

```
Dim SmartServer As iLON_SoapCalls = New iLON_SoapCalls
SmartServer.BindClientToSmartServer()

Try

' -------------- CREATING A DATA LOGGER --------------

'Create an xSelect object and then specify the filter to be used

Dim xSelect As New iLON_SmartServer.E_xSelect()
xSelect.xSelect = "//Item[@xsi:type=""LON_Fb_Cfg""][contains(UCPTname,""Data Logger"")][UCPThidden = ""1""]"

'Create an ItemColl that stores objects returned by List()function that takes an xSelect object

Dim ItemColl As iLON_SmartServer.Item_Coll = SmartServer._iLON.List(xSelect)

'Create an ItemCfgColl that stores the objects to be returned by a Get() function that takes
 'the ItemColl returned by the List()

 ItemColl.xSelect = "//Item[@xsi:type=""LON_Fb_Cfg""]"
 Dim ItemCfgColl__1 As iLON_SmartServer.Item_CfgColl = SmartServer._iLON.Get(ItemColl)

'check that there are obejcts in the ItemCfgColl

If ItemCfgColl__1.UCPTfaultCount > 0 Then
    PrintGetError(ItemCfgColl__1)
Else

    'Create LON_Fb_Cfg item
    ItemCfgColl__1.Item(0).UCPThidden = 0
    ItemCfgColl__1.Item(0).UCPTname = "Net/LON/iLON App/myDataLogger"
    Dim ItemColl_SetReturn As iLON_SmartServer.Item_Coll =
       SmartServer._iLON.Set(ItemCfgColl__1)

    'create new Data Logger from existing one
    Dim myDataLogger As iLON_SmartServer.UFPTdataLogger_Cfg =
       New iLON_SmartServer.UFPTdataLogger_Cfg()
    myDataLogger.UCPTname = "Net/LON/iLON App/myDataLogger"
    myDataLogger.UCPTannotation = "#8000010128000000[4].UFPTdataLogger"
    myDataLogger.UCPTlogFileName = "Net/LON/iLON App/myDataLogger.csv"
    myDataLogger.UCPTlogSize = 100
    myDataLogger.UCPTlogLevelAlarm = 50

    myDataLogger.UCPTlogType = New iLON_SmartServer.E_LonString()
    myDataLogger.UCPTlogType.Value = "LT_HISTORICAL"
    myDataLogger.UCPTlogType.LonFormat = "UCPTlogType"

    myDataLogger.UCPTlogFormat = New iLON_SmartServer.E_LonString()
    myDataLogger.UCPTlogFormat.Value = "LF_TEXT"
    myDataLogger.UCPTlogFormat.LonFormat = "UCPTlogFormat"

    'create DP reference array to store data points by new Data Logger
    myDataLogger.DataPoint = New iLON_SmartServer.E_DpRef(1) {}

    'speficy data points to be logged by new Data Logger

    Dim dataPointRef1 As New iLON_SmartServer.UFPTdataLogger_DpRef()
    dataPointRef1.UCPTname = "Net/LON/iLON App/Digital Output 2/nviClaValue_2"
    dataPointRef1.UCPTformatDescription = "#0000000000000000[0].SNVT_switch"
    dataPointRef1.UCPTpollRate = 60
    dataPointRef1.dpType = "Input"

    Dim dataPointRef2 As New iLON_SmartServer.UFPTdataLogger_DpRef()
    dataPointRef2.UCPTname = "Net/LON/iLON App/Digital Output 1/nviClaValue_1"
    dataPointRef2.UCPTformatDescription = "#0000000000000000[0].SNVT_switch"
    dataPointRef2.UCPTpollRate = 60
    dataPointRef2.dpType = "Input"

    'store data points in DP reference array
    myDataLogger.DataPoint(0) = dataPointRef1
```

```
            myDataLogger.DataPoint(1) = dataPointRef2

            'call Set function
            Dim itemCfgColl__2 As New iLON_SmartServer.Item_CfgColl()
            itemCfgColl__2.Item = New iLON_SmartServer.Item_Cfg(0) {}
            itemCfgColl__2.Item(0) = myDataLogger

            Dim ItemColl_Set_DataLogger_Return As iLON_SmartServer.Item_Coll =
               SmartServer._iLON.Set(itemCfgColl__2)

            If ItemColl_Set_DataLogger_Return.UCPTfaultCount > 0 Then
                Exit Sub
            Else

                Dim newDataLogger As iLON_SmartServer.Item = ItemColl_Set_DataLogger_Return.Item(0)
                Console.WriteLine("New Data Logger = " & newDataLogger.UCPTname)
            End If
        End If

        Console.ReadLine()

      Finally
       SmartServer.CloseBindingToSmartServer()

     End Try
    End Sub
End Module
```

21.2.2.2 Reading a Data Logger

This VB console example reads and prints out the last 10 entries for one of the two data points recorded by the new data logger you created in the previous section, *Creating a Data Logger*. For more information on the data logger properties used in this example, see section 5.3.4, *Using the Read Function on a Data Logger*.

```
Module ReadDataLogModule

    Private Sub PrintGetError(ByVal ItemColl As iLON_SmartServer.Item_Coll)
        ' print out error and exit
        Console.Out.WriteLine("An error occurred:")
        For j As Integer = 0 To ItemColl.Item.Length - 1
            If ItemColl.Item(j).fault IsNot Nothing Then
                Console.Out.WriteLine((("Item: " & ItemColl.Item(j).UCPTname & ", fault code: ") +
                ItemColl.Item(j).fault.faultcode.Value & ", fault string: ") +
                ItemColl.Item(j).fault.faultstring)
            End If
        Next
    End Sub

    Public Sub Main()

        Dim SmartServer As iLON_SoapCalls = New iLON_SoapCalls
        SmartServer.BindClientToSmartServer()

       Try

        ' -------------- READING A DATA LOGGER --------------

        'Create an xSelect object and then specify the filter to be used

        Dim xSelect As New iLON_SmartServer.E_xSelect()
       xSelect.xSelect = "//Item[UCPTname = ""Net/LON/iLON App/myDataLogger""]"

        'Create an ItemColl that stores objects returned by List()function that takes an xSelect object

        Dim ItemColl As iLON_SmartServer.Item_Coll = SmartServer._iLON.List(xSelect)

        'check that there are obejcts in the ItemColl

        If ItemColl.UCPTfaultCount > 0 Then
```

```
            PrintGetError(ItemColl)
        Else
            Dim myDataLogger As iLON_SmartServer.Item = ItemColl.Item(0)
            Console.WriteLine("Data Logger = " & myDataLogger.UCPTname & vbCr & vbLf & vbCr & vbLf)
        End If

        'we use an xSelect to read only the last 10 records in the Data Logger for one data point
        ItemColl.xSelect =
           "//Item[UCPTpointName=""Net/LON/iLON App/Digital Output 1/nviClaValue_1""][position()>=last()-10]"

        ' Read Data Logger
        Dim dataLogger As iLON_SmartServer.Item_DataColl = SmartServer._iLON.Read(ItemColl)

        For i As Integer = 0 To dataLogger.Item.Length - 1
                Dim dataLoggerDataCheck As iLON_SmartServer.UFPTdataLogger_Data =
                   TryCast(dataLogger.Item(i), iLON_SmartServer.UFPTdataLogger_Data)

                If dataLoggerDataCheck IsNot Nothing Then
                    Dim dataLoggerData As iLON_SmartServer.UFPTdataLogger_Data =
                      DirectCast(dataLogger.Item(i), iLON_SmartServer.UFPTdataLogger_Data)
            Console.Out.WriteLine(((dataLoggerData.UCPTname & " was ") +
             dataLoggerData.UCPTvalue(0).Value & " at ") + dataLoggerData.UCPTlastUpdate & vbCr & vbLf)
        Next
         Console.ReadLine()

      Finally
       SmartServer.CloseBindingToSmartServer()

      End Try
    End Sub
End Module
```

21.2.3 Creating a Scheduler and Calendar in Visual Basic.NET

This VB console example creates a Scheduler and Calendar for hypothetically controlling the lighting and heating of a store. You can execute this code after you have referenced and inherited from the SmartServer WSDL as described in section 20.1, and instantiated and initialized the Web service client as described in section 20.2.

The example creates a new Scheduler from an existing uninstantiated (hidden) Scheduler on the SmartServer. It creates separate daily schedules for weekdays, Saturdays, and Sundays, and it specifies that the scheduler turn on and off the SmartServer's digital relay outputs (the Net/LON/iLON App/Digital Output 1/nviClaValue_1 and Net/LON/iLON App/Digital Output 2/nviClaValue_2 data points) at specific times based on the day of the week. The example then creates an exception that keeps the lighting and heating off on holidays.

After creating the Scheduler, this example either gets the Calendar on the SmartServer if it has already been instantiated or creates a new Calendar. The example then specifies the dates of the holidays for the exception created in the Scheduler, and it specifies over how many years the holiday exceptions are to occur.

For more information on the Scheduler and Calendar properties set in this example, see section 9.3.2, *Using the Get Function a Scheduler* and section 10.3.2, *Using the Get Function a Calendar*, respectively.

Note: The <UCPTexceptionName> property is the unique identifier for exceptions defined in the Scheduler and Calendar. This means that the <UCPTexceptionName> property of new exceptions you create must be unique to the Calendar; otherwise, the exception you create will overwrite an existing exception. To prevent overwriting an existing exception, you can loop through the existing exceptions on the Calendar and check whether the <UCPTexceptionName> property of the exception you are creating matches that of any existing exceptions. This example assumes that your SmartServer has been set to its factory default settings and therefore does not perform this check.

```
Module SchedulerModule

    Private Sub PrintGetError(ByVal ItemCfgColl As iLON_SmartServer.Item_CfgColl)
```

```
        ' print out error and exit
        Console.Out.WriteLine("An error occurred:")
        For j As Integer = 0 To ItemCfgColl.Item.Length - 1
            If ItemCfgColl.Item(j).fault IsNot Nothing Then
                Console.Out.WriteLine((("Item: " & ItemCfgColl.Item(j).UCPTname & ", fault code: ") +
                   ItemCfgColl.Item(j).fault.faultcode.Value & ", fault string: ") +
                   ItemCfgColl.Item(j).fault.faultstring)
            End If
        Next
    End Sub

    Private Sub PrintGetError(ByVal ItemColl As iLON_SmartServer.Item_Coll)
        ' print out error and exit
        Console.Out.WriteLine("An error occurred:")
        For j As Integer = 0 To ItemColl.Item.Length - 1
            If ItemColl.Item(j).fault IsNot Nothing Then
                Console.Out.WriteLine((("Item: " & ItemColl.Item(j).UCPTname & ", fault code: ") +
                   ItemColl.Item(j).fault.faultcode.Value & ", fault string: ") +
                   ItemColl.Item(j).fault.faultstring)
            End If
        Next
    End Sub

    Public Sub Main()

        Dim SmartServer As iLON_SoapCalls = New iLON_SoapCalls
        SmartServer.BindClientToSmartServer()

       Try

        ' -------------- CREATING A SCHEDULER --------------

        'Create an xSelect object and then specify the filter to be used

        Dim xSelect As New iLON_SmartServer.E_xSelect()
        xSelect.xSelect =
         "//Item[@xsi:type=""LON_Fb_Cfg""][contains(UCPTname,""Scheduler"")][UCPThidden = ""1""]"

        'Create an ItemColl that stores objects returned by List()function that takes an xSelect object

        Dim ItemColl As iLON_SmartServer.Item_Coll = SmartServer._iLON.List(xSelect)

        'Create an ItemCfgColl that stores the objects to be returned by a Get() function that takes
         'the ItemColl returned by the List()

       ItemColl.xSelect = "//Item[@xsi:type=""LON_Fb_Cfg""]"
        Dim ItemCfgColl__1 As iLON_SmartServer.Item_CfgColl = SmartServer._iLON.Get(ItemColl)

        'check that there are obejcts in the ItemCfgColl

        If ItemCfgColl__1.UCPTfaultCount > 0 Then
            PrintGetError(ItemCfgColl__1)
        Else


            'Create LON_Fb_Cfg item
            ItemCfgColl__1.Item(0).UCPThidden = 0
            ItemCfgColl__1.Item(0).UCPTname = "Net/LON/iLON App/myScheduler"
            Dim ItemColl_SetReturn As iLON_SmartServer.Item_Coll = SmartServer._iLON.Set(ItemCfgColl__1)

            'create new Scheduler from existing one
            Dim myScheduler As New iLON_SmartServer.UFPTscheduler_Cfg()
            myScheduler.UCPTname = "Net/LON/iLON App/myScheduler"
            myScheduler.UCPTannotation = "#8000010128000000[4].UFPTscheduler"

            'create DP reference array to store data points controlled by new Scheduler
            myScheduler.DataPoint = New iLON_SmartServer.E_DpRef(1) {}

            'speficy data points to be controlled by new Scheduler

            Dim dataPointRef1 As New iLON_SmartServer.UFPTscheduler_DpRef()
```

```
dataPointRef1.UCPTname = "Net/LON/iLON App/Digital Output 1/nviClaValue_1"
dataPointRef1.UCPTformatDescription = "#0000000000000000[0].SNVT_switch"
dataPointRef1.SCPTdelayTime = 0
dataPointRef1.SCPTdelayTimeSpecified = True
dataPointRef1.dpType = "Output"

Dim dataPointRef2 As New iLON_SmartServer.UFPTscheduler_DpRef()
dataPointRef2.UCPTname = "Net/LON/iLON App/Digital Output 2/nviClaValue_2"
dataPointRef2.UCPTformatDescription = "#0000000000000000[0].SNVT_switch"
dataPointRef2.SCPTdelayTime = 0
dataPointRef2.SCPTdelayTimeSpecified = True
dataPointRef2.dpType = "Output"

'store data points in DP reference array
myScheduler.DataPoint(0) = dataPointRef1
myScheduler.DataPoint(1) = dataPointRef2

'set range of dates in which Scheduler is effective
Dim effectivePeriod As New iLON_SmartServer.UFPTscheduler_CfgEffectivePeriod()
effectivePeriod.StartDate = New DateTime(2009, 6, 8)
effectivePeriod.EndDate = New DateTime(2020, 12, 31)
effectivePeriod.StartDateSpecified = True
effectivePeriod.EndDateSpecified = True
myScheduler.ScheduleEffectivePeriod = effectivePeriod

'create daily schedule for weekdays
Dim dayBasedSchedule_weekdays As New iLON_SmartServer.UFPTscheduler_CfgDayBased()
dayBasedSchedule_weekdays.UCPTindex = 0
dayBasedSchedule_weekdays.UCPTindexSpecified = True
dayBasedSchedule_weekdays.UCPTdescription = "Weekday"
dayBasedSchedule_weekdays.UCPTpriority = 255

'create events for weekday schedule

dayBasedSchedule_weekdays.[Event] = New iLON_SmartServer.UFPTscheduler_CfgEvent(1) {}
dayBasedSchedule_weekdays.[Event](0) = New iLON_SmartServer.UFPTscheduler_CfgEvent()
dayBasedSchedule_weekdays.[Event](1) = New iLON_SmartServer.UFPTscheduler_CfgEvent()

'---create ON event----
Dim onEvent As New iLON_SmartServer.UFPTscheduler_CfgEvent()
onEvent.UCPTindex = 0
onEvent.UCPTindexSpecified = True
onEvent.UCPTtime = New DateTime(2009, 6, 8, 10, 0, 0)

onEvent.UCPTvalue = New iLON_SmartServer.E_LonString(0) {}
onEvent.UCPTvalue(0) = New iLON_SmartServer.E_LonString()
onEvent.UCPTvalue(0).Value = "ON"
onEvent.UCPTvalue(0).LonFormat = "UCPTvalueDef"

dayBasedSchedule_weekdays.[Event](0) = onEvent

'---create OFF event---
Dim offEvent As New iLON_SmartServer.UFPTscheduler_CfgEvent()
offEvent.UCPTindex = 1
offEvent.UCPTindexSpecified = True
offEvent.UCPTtime = New DateTime(2009, 6, 8, 21, 0, 0)

offEvent.UCPTvalue = New iLON_SmartServer.E_LonString(0) {}
offEvent.UCPTvalue(0) = New iLON_SmartServer.E_LonString()
offEvent.UCPTvalue(0).Value = "OFF"
offEvent.UCPTvalue(0).LonFormat = "UCPTvalueDef"

dayBasedSchedule_weekdays.[Event](1) = offEvent

'set Monday--Friday as the days in this daily schedule
Dim mon_to_fri As New iLON_SmartServer.UFPTscheduler_CfgDayBasedWeekdays()
mon_to_fri.UCPTmonday = 1
mon_to_fri.UCPTtuesday = 1
mon_to_fri.UCPTwednesday = 1
mon_to_fri.UCPTthursday = 1
mon_to_fri.UCPTfriday = 1
```

```
mon_to_fri.UCPTsaturday = 0
mon_to_fri.UCPTsunday = 0

dayBasedSchedule_weekdays.Weekdays = mon_to_fri

'create daily schedule for Saturdays

Dim dayBasedSchedule_Sat As New iLON_SmartServer.UFPTscheduler_CfgDayBased()
dayBasedSchedule_Sat.UCPTindex = 1
dayBasedSchedule_Sat.UCPTindexSpecified = True
dayBasedSchedule_Sat.UCPTdescription = "Saturday"
dayBasedSchedule_Sat.UCPTpriority = 255

'create events for Saturday schedule

dayBasedSchedule_Sat.[Event] = New iLON_SmartServer.UFPTscheduler_CfgEvent(1) {}
dayBasedSchedule_Sat.[Event](0) = New iLON_SmartServer.UFPTscheduler_CfgEvent()
dayBasedSchedule_Sat.[Event](1) = New iLON_SmartServer.UFPTscheduler_CfgEvent()

'---create ON event----
Dim onEvent_Sat As New iLON_SmartServer.UFPTscheduler_CfgEvent()
onEvent_Sat.UCPTindex = 0
onEvent_Sat.UCPTindexSpecified = True
onEvent_Sat.UCPTtime = New DateTime(2009, 6, 8, 10, 0, 0)

onEvent_Sat.UCPTvalue = New iLON_SmartServer.E_LonString(0) {}
onEvent_Sat.UCPTvalue(0) = New iLON_SmartServer.E_LonString()
onEvent_Sat.UCPTvalue(0).Value = "ON"
onEvent_Sat.UCPTvalue(0).LonFormat = "UCPTvalueDef"

dayBasedSchedule_Sat.[Event](0) = onEvent_Sat

'---create OFF event---
Dim offEvent_Sat As New iLON_SmartServer.UFPTscheduler_CfgEvent()
offEvent_Sat.UCPTindex = 1
offEvent_Sat.UCPTindexSpecified = True
offEvent_Sat.UCPTtime = New DateTime(2009, 6, 8, 19, 0, 0)

offEvent_Sat.UCPTvalue = New iLON_SmartServer.E_LonString(0) {}
offEvent_Sat.UCPTvalue(0) = New iLON_SmartServer.E_LonString()
offEvent_Sat.UCPTvalue(0).Value = "OFF"
offEvent_Sat.UCPTvalue(0).LonFormat = "UCPTvalueDef"

dayBasedSchedule_Sat.[Event](1) = offEvent_Sat

'set Saturday as only day in this daily schedule
Dim sat As New iLON_SmartServer.UFPTscheduler_CfgDayBasedWeekdays()
sat.UCPTsaturday = 1

sat.UCPTsunday = 0
sat.UCPTmonday = 0
sat.UCPTtuesday = 0
sat.UCPTwednesday = 0
sat.UCPTthursday = 0
sat.UCPTfriday = 0

dayBasedSchedule_Sat.Weekdays = sat

'create daily schedule for Sundays
Dim dayBasedSchedule_Sun As New iLON_SmartServer.UFPTscheduler_CfgDayBased()
dayBasedSchedule_Sun.UCPTindex = 2
dayBasedSchedule_Sun.UCPTindexSpecified = True
dayBasedSchedule_Sun.UCPTdescription = "Sunday"
dayBasedSchedule_Sun.UCPTpriority = 255

'create events for Sunday Schedule

dayBasedSchedule_Sun.[Event] = New iLON_SmartServer.UFPTscheduler_CfgEvent(1) {}
dayBasedSchedule_Sun.[Event](0) = New iLON_SmartServer.UFPTscheduler_CfgEvent()
dayBasedSchedule_Sun.[Event](1) = New iLON_SmartServer.UFPTscheduler_CfgEvent()
```

```
'---create ON event----
Dim onEvent_Sun As New iLON_SmartServer.UFPTscheduler_CfgEvent()
onEvent_Sun.UCPTindex = 0
onEvent_Sun.UCPTindexSpecified = True
onEvent_Sun.UCPTtime = New DateTime(2009, 6, 8, 12, 0, 0)

onEvent_Sun.UCPTvalue = New iLON_SmartServer.E_LonString(0) {}
onEvent_Sun.UCPTvalue(0) = New iLON_SmartServer.E_LonString()
onEvent_Sun.UCPTvalue(0).Value = "ON"
onEvent_Sun.UCPTvalue(0).LonFormat = "UCPTvalueDef"

dayBasedSchedule_Sun.[Event](0) = onEvent_Sun

'---create OFF event---
Dim offEvent_Sun As New iLON_SmartServer.UFPTscheduler_CfgEvent()
offEvent_Sun.UCPTindex = 1
offEvent_Sun.UCPTindexSpecified = True
offEvent_Sun.UCPTtime = New DateTime(2009, 6, 8, 18, 0, 0)

offEvent_Sun.UCPTvalue = New iLON_SmartServer.E_LonString(0) {}
offEvent_Sun.UCPTvalue(0) = New iLON_SmartServer.E_LonString()
offEvent_Sun.UCPTvalue(0).Value = "OFF"
offEvent_Sun.UCPTvalue(0).LonFormat = "UCPTvalueDef"

dayBasedSchedule_Sun.[Event](1) = offEvent_Sun

'set Sunday as only day in this daily schedule
Dim sun As New iLON_SmartServer.UFPTscheduler_CfgDayBasedWeekdays()
sun.UCPTsunday = 1

sun.UCPTsaturday = 0
sun.UCPTmonday = 0
sun.UCPTtuesday = 0
sun.UCPTwednesday = 0
sun.UCPTthursday = 0
sun.UCPTfriday = 0

dayBasedSchedule_Sun.Weekdays = sun

'store daily schedules we created in a DayBased[]

myScheduler.DayBased = New iLON_SmartServer.UFPTscheduler_CfgDayBased(2) {}

myScheduler.DayBased(0) = New iLON_SmartServer.UFPTscheduler_CfgDayBased()
myScheduler.DayBased(0) = dayBasedSchedule_weekdays

myScheduler.DayBased(1) = New iLON_SmartServer.UFPTscheduler_CfgDayBased()
myScheduler.DayBased(1) = dayBasedSchedule_Sat

myScheduler.DayBased(2) = New iLON_SmartServer.UFPTscheduler_CfgDayBased()
myScheduler.DayBased(2) = dayBasedSchedule_Sun

'create a date-based schedule (an exception) for Holidays

'*** NOTE: You must use Calendar application to spcify the dates in which this alterntate
'schedule is applicable***

Dim holidays As New iLON_SmartServer.UFPTscheduler_CfgDateBased()
holidays.UCPTindex = 0
holidays.UCPTindexSpecified = True
holidays.UCPTpriority = 250

'create events for Holiday schedule

holidays.[Event] = New iLON_SmartServer.UFPTscheduler_CfgEvent(2) {}
holidays.[Event](0) = New iLON_SmartServer.UFPTscheduler_CfgEvent()
holidays.[Event](1) = New iLON_SmartServer.UFPTscheduler_CfgEvent()
holidays.[Event](2) = New iLON_SmartServer.UFPTscheduler_CfgEvent()

'create LOCK event at 00:00 for Holiday schedule
```

```
Dim lockEvent_holiday As New iLON_SmartServer.UFPTscheduler_CfgEvent()
lockEvent_holiday.UCPTindex = 0
lockEvent_holiday.UCPTindexSpecified = True
lockEvent_holiday.UCPTtime = New DateTime(2009, 6, 8, 0, 0, 0)

lockEvent_holiday.UCPTeventType = New iLON_SmartServer.E_LonString()
lockEvent_holiday.UCPTeventType.Value = "ET_LOCK"
lockEvent_holiday.UCPTeventType.LonFormat = "UCPTeventType"

holidays.[Event](0) = lockEvent_holiday

'create ON event for Holiday schedule
Dim onEvent_holiday As New iLON_SmartServer.UFPTscheduler_CfgEvent()
onEvent_holiday.UCPTindex = 1
onEvent_holiday.UCPTindexSpecified = True
onEvent_holiday.UCPTtime = New DateTime(2009, 6, 8, 12, 0, 0)

onEvent_holiday.UCPTeventType = New iLON_SmartServer.E_LonString()
onEvent_holiday.UCPTeventType.Value = "ET_NUL"
onEvent_holiday.UCPTeventType.LonFormat = "UCPTeventType"

onEvent_holiday.UCPTvalue = New iLON_SmartServer.E_LonString(0) {}
onEvent_holiday.UCPTvalue(0) = New iLON_SmartServer.E_LonString()
onEvent_holiday.UCPTvalue(0).Value = "ON"
onEvent_holiday.UCPTvalue(0).LonFormat = "UCPTvalueDef"

holidays.[Event](1) = onEvent_holiday

'create OFF event for Holiday schedule

Dim offEvent_holiday As New iLON_SmartServer.UFPTscheduler_CfgEvent()
offEvent_holiday.UCPTindex = 2
offEvent_holiday.UCPTindexSpecified = True
offEvent_holiday.UCPTtime = New DateTime(2009, 6, 8, 18, 0, 0)

offEvent_holiday.UCPTeventType = New iLON_SmartServer.E_LonString()
offEvent_holiday.UCPTeventType.Value = "ET_NUL"
offEvent_holiday.UCPTeventType.LonFormat = "UCPTeventType"

offEvent_holiday.UCPTvalue = New iLON_SmartServer.E_LonString(0) {}
offEvent_holiday.UCPTvalue(0) = New iLON_SmartServer.E_LonString()
offEvent_holiday.UCPTvalue(0).Value = "OFF"
offEvent_holiday.UCPTvalue(0).LonFormat = "UCPTvalueDef"

holidays.[Event](2) = offEvent_holiday

'create Exception item
holidays.Exception = New iLON_SmartServer.UFPTscheduler_CfgDateBasedException(0) {}
holidays.Exception(0) = New iLON_SmartServer.UFPTscheduler_CfgDateBasedException()
holidays.Exception(0).UCPTexceptionName = "Holidays"
holidays.Exception(0).UCPTindex = 0
holidays.Exception(0).UCPTindexSpecified = True

'store date-based (exception) schedule we created in a DateBased[]

myScheduler.DateBased = New iLON_SmartServer.UFPTscheduler_CfgDateBased(0) {}

myScheduler.DateBased(0) = New iLON_SmartServer.UFPTscheduler_CfgDateBased()
myScheduler.DateBased(0) = holidays

'call Set function
Dim itemCfgColl__2 As New iLON_SmartServer.Item_CfgColl()
itemCfgColl__2.Item = New iLON_SmartServer.Item_Cfg(0) {}
itemCfgColl__2.Item(0) = myScheduler

Dim ItemColl_Set_Scheduler_Return As iLON_SmartServer.Item_Coll =
   SmartServer._iLON.Set(itemCfgColl__2)

If ItemColl_Set_Scheduler_Return.UCPTfaultCount > 0 Then
    PrintGetError(ItemColl_Set_Scheduler_Return)
Else
```

```
        Dim newScheduler As iLON_SmartServer.Item = ItemColl_Set_Scheduler_Return.Item(0)
        Console.WriteLine("New Scheduler = " & newScheduler.UCPTname)
    End If
End If

' -------------- CREATING A CALENDAR --------------

Create a new UFPTcalendar_Cfg item
 Dim myCalendar As New iLON_SmartServer.UFPTcalendar_Cfg()
 myCalendar.UCPTname = "Net/LON/iLON App/myCalendar"
 myCalendar.UCPTannotation = "#80000101280000000[4].UFPTcalendar"

End Try

'Configure the Calendar
Dim effectivePeriod_calendar As New iLON_SmartServer.UFPTscheduler_CfgEffectivePeriod()
effectivePeriod_calendar.StartDate = New DateTime(2009, 6, 8)
effectivePeriod_calendar.EndDate = New DateTime(2020, 12, 31)
effectivePeriod_calendar.StartDateSpecified = True
effectivePeriod_calendar.EndDateSpecified = True
myCalendar.ScheduleEffectivePeriod = effectivePeriod_calendar

'create an exception
myCalendar.Exception = New iLON_SmartServer.UFPTcalendar_CfgException(0) {}
myCalendar.Exception(0) = New iLON_SmartServer.UFPTcalendar_CfgException()

myCalendar.Exception(0).UCPTexceptionName = "Holidays"
myCalendar.Exception(0).UCPTaliasName = "Holidays"
myCalendar.Exception(0).UCPTindex = 0
myCalendar.Exception(0).UCPTindexSpecified = True
myCalendar.Exception(0).UCPTtemporary = 0
myCalendar.Exception(0).UCPTtemporarySpecified = True
myCalendar.Exception(0).UCPTmaxClient = 1
myCalendar.Exception(0).UCPTmaxClientSpecified = True

myCalendar.Exception(0).Client = New iLON_SmartServer.UFPTcalendar_CfgExceptionClient(0) {}
myCalendar.Exception(0).Client(0) = New iLON_SmartServer.UFPTcalendar_CfgExceptionClient()
myCalendar.Exception(0).Client(0).UCPTname = "Net/LON/iLON App/myScheduler"
 myCalendar.Exception(0).Client(0).UCPTservicePath = New iLON_SmartServer.E_Path()
myCalendar.Exception(0).Client(0).UCPTservicePath.Value = ""

'create exception dates
myCalendar.Exception(0).Schedule = New iLON_SmartServer.UFPTcalendar_CfgExceptionSchedule(3) {}
myCalendar.Exception(0).Schedule(0) = New iLON_SmartServer.UFPTcalendar_CfgExceptionSchedule()
myCalendar.Exception(0).Schedule(1) = New iLON_SmartServer.UFPTcalendar_CfgExceptionSchedule()
myCalendar.Exception(0).Schedule(2) = New iLON_SmartServer.UFPTcalendar_CfgExceptionSchedule()
myCalendar.Exception(0).Schedule(3) = New iLON_SmartServer.UFPTcalendar_CfgExceptionSchedule()

'create 4th of July exception
'==========================

myCalendar.Exception(0).Schedule(0).UCPTschedMonth = New iLON_SmartServer.E_LonString()
myCalendar.Exception(0).Schedule(0).UCPTschedMonth.LonFormat = "UCPTschedMonth"
myCalendar.Exception(0).Schedule(0).UCPTschedMonth.Value = "MN_JUL"

myCalendar.Exception(0).Schedule(0).UCPTschedDay = New iLON_SmartServer.E_LonString()
myCalendar.Exception(0).Schedule(0).UCPTschedDay.LonFormat = "UCPTschedDay"
myCalendar.Exception(0).Schedule(0).UCPTschedDay.Value = "DM_DAY_4"

'set start date
myCalendar.Exception(0).Schedule(0).StartDate = New iLON_SmartServer.UFPTcalendar_CfgESDate()
myCalendar.Exception(0).Schedule(0).StartDate.UCPTdate = New DateTime(2009, 6, 8)

'set end date
myCalendar.Exception(0).Schedule(0).EndDate = New iLON_SmartServer.UFPTcalendar_CfgESDate()
myCalendar.Exception(0).Schedule(0).EndDate.UCPTdate = New DateTime(2020, 12, 31)

'create Labor Day exception
'==========================
```

```
myCalendar.Exception(0).Schedule(1).UCPTschedMonth = New iLON_SmartServer.E_LonString()
myCalendar.Exception(0).Schedule(1).UCPTschedMonth.LonFormat = "UCPTschedMonth"
myCalendar.Exception(0).Schedule(1).UCPTschedMonth.Value = "MN_SEP"

myCalendar.Exception(0).Schedule(1).UCPTschedDay = New iLON_SmartServer.E_LonString()
myCalendar.Exception(0).Schedule(1).UCPTschedDay.LonFormat = "UCPTschedDay"
myCalendar.Exception(0).Schedule(1).UCPTschedDay.Value = "DM_FIRST_MON"

'set start date
myCalendar.Exception(0).Schedule(1).StartDate = New iLON_SmartServer.UFPTcalendar_CfgESDate()
myCalendar.Exception(0).Schedule(1).StartDate.UCPTdate = New DateTime(2009, 6, 8)

'set end date
myCalendar.Exception(0).Schedule(1).EndDate = New iLON_SmartServer.UFPTcalendar_CfgESDate()
myCalendar.Exception(0).Schedule(1).EndDate.UCPTdate = New DateTime(2020, 12, 31)

'create Thanksgiving exception
'==========================

myCalendar.Exception(0).Schedule(2).UCPTschedMonth = New iLON_SmartServer.E_LonString()
myCalendar.Exception(0).Schedule(2).UCPTschedMonth.LonFormat = "UCPTschedMonth"
myCalendar.Exception(0).Schedule(2).UCPTschedMonth.Value = "MN_NOV"

myCalendar.Exception(0).Schedule(2).UCPTschedDay = New iLON_SmartServer.E_LonString()
myCalendar.Exception(0).Schedule(2).UCPTschedDay.LonFormat = "UCPTschedDay"
myCalendar.Exception(0).Schedule(2).UCPTschedDay.Value = "DM_FOURTH_THU"

'set start date
myCalendar.Exception(0).Schedule(2).StartDate = New iLON_SmartServer.UFPTcalendar_CfgESDate()
myCalendar.Exception(0).Schedule(2).StartDate.UCPTdate = New DateTime(2009, 6, 8)

'set end date
myCalendar.Exception(0).Schedule(2).EndDate = New iLON_SmartServer.UFPTcalendar_CfgESDate()
myCalendar.Exception(0).Schedule(2).EndDate.UCPTdate = New DateTime(2020, 12, 31)

'Create Christmas exception
'==========================

myCalendar.Exception(0).Schedule(3).UCPTschedMonth = New iLON_SmartServer.E_LonString()
myCalendar.Exception(0).Schedule(3).UCPTschedMonth.LonFormat = "UCPTschedMonth"
myCalendar.Exception(0).Schedule(3).UCPTschedMonth.Value = "MN_DEC"

myCalendar.Exception(0).Schedule(3).UCPTschedDay = New iLON_SmartServer.E_LonString()
myCalendar.Exception(0).Schedule(3).UCPTschedDay.LonFormat = "UCPTschedDay"
myCalendar.Exception(0).Schedule(3).UCPTschedDay.Value = "DM_DAY_25"

'set start date
myCalendar.Exception(0).Schedule(3).StartDate = New iLON_SmartServer.UFPTcalendar_CfgESDate()
myCalendar.Exception(0).Schedule(3).StartDate.UCPTdate = New DateTime(2009, 6, 8)

'set end date
myCalendar.Exception(0).Schedule(3).EndDate = New iLON_SmartServer.UFPTcalendar_CfgESDate()
myCalendar.Exception(0).Schedule(3).EndDate.UCPTdate = New DateTime(2020, 12, 31)

'call Set function
Dim itemCfgColl_Calendar__4 As New iLON_SmartServer.Item_CfgColl()
itemCfgColl_Calendar__4.Item = New iLON_SmartServer.Item_Cfg(0) {}
itemCfgColl_Calendar__4.Item(0) = myCalendar


Dim ItemColl_Set_Calendar_Return As iLON_SmartServer.Item_Coll =
   SmartServer._iLON.Set(itemCfgColl_Calendar__4)

If ItemColl_Set_Calendar_Return.UCPTfaultCount > 0 Then
    PrintGetError(ItemColl_Set_Calendar_Return)
Else

    Dim newCalendar As iLON_SmartServer.Item = ItemColl_Set_Calendar_Return.Item(0)
    Console.WriteLine("Calendar used for this Scheduler is " & newCalendar.UCPTname)
End If
```

```
        Console.ReadLine()
      Finally
       SmartServer.CloseBindingToSmartServer()
      End Try
    End Sub
End Module
```

21.2.4 Creating and Installing a LONWORKS Device in Visual Basic.NET

This VB console example creates two LONWORKS devices, and then it commissions the devices, starts the devices' applications, and gets the devices' templates (to display the devices' functional blocks and data points in the SmartServer Web interface). The example then prints out the names and statuses of the devices that have been installed. Note that you need to replace the values of the <UCPTname>, <UCPTuniqueID>, <UCPTprogramID>, and <UCPTurlTemplate> properties provided in this example with those of the devices you are creating and installing.

You can execute this code after you have referenced and inherited from the SmartServer WSDL as described in section 20.1, and instantiated and initialized the Web service client as described in section 20.2. You must also upload the device interface (XIF) files of the devices you are creating to the root/LonWorks/import folder on the SmartServer flash disk.

For more information on the LONWORKS device properties set in this example, see section 14.3.2, *Using the Get Function on a LonWorks Device*. For more information on the network management commands issues in this example, see section 14.3.3.1, *Issuing Network Management Commands*.

```
Module InstallDeviceModule

    'Function required for converting device Neuron IDs and program IDs to a byte[]
    Public Function HexStringToArray(ByVal str As String) As Byte()
        Dim nLen As Integer = str.Length / 2
        Dim arr As Byte() = New Byte(nLen - 1) {}

        For i As Integer = 0 To nLen - 1
            Dim strByte As String = str.Substring(i * 2, 2)
            arr(i) = [Byte].Parse(strByte, System.Globalization.NumberStyles.HexNumber)
        Next
        Return arr
    End Function

    Public Sub Main()

        Dim SmartServer As iLON_SoapCalls = New iLON_SoapCalls
        SmartServer.BindClientToSmartServer()

       Try

        ' -------------- CREATING LONWORKS DEVICES --------------

        'Create a new LON_Device_Cfg Item

        Dim my_LON_Device1 As New iLON_SmartServer.LON_Device_Cfg()
        Dim my_LON_Device2 As New iLON_SmartServer.LON_Device_Cfg()

        'Create an ItemCfgColl to store the LON Devices we just created

        Dim ItemCfgColl As New iLON_SmartServer.Item_CfgColl()
        ItemCfgColl.Item = New iLON_SmartServer.Item_Cfg(1) {}
        ItemCfgColl.Item(0) = my_LON_Device1
        ItemCfgColl.Item(1) = my_LON_Device2

        '=====CREATING AND INSTALLING LON DEVICE #1==================

        ' specify properties of new LON Device #1
        my_LON_Device1.UCPTname = "Net/LON/DIO-1"
        my_LON_Device1.UCPTlocal = 0
        my_LON_Device1.UCPTuniqueId = HexStringToArray("00a145791500")
```

```
my_LON_Device1.UCPTprogramId = HexStringToArray("80000105288a0403")
my_LON_Device1.UCPTurlTemplate = "/root/lonWorks/Import/Echelon/LonPoint/Version3/dio-10v3.XIF"

my_LON_Device1.UCPTcommissionStatus = New iLON_SmartServer.E_LonString()
my_LON_Device1.UCPTcommissionStatus.Value = "COMMISSIONED"
my_LON_Device1.UCPTapplicationStatus = New iLON_SmartServer.E_LonString()
my_LON_Device1.UCPTapplicationStatus.Value = "APP_RUNNING"

'create a command array to store device commands to be sent
my_LON_Device1.Command = New iLON_SmartServer.LON_Device_CfgCommand(2) {}

'commission device
my_LON_Device1.Command(0) = New iLON_SmartServer.LON_Device_CfgCommand()
my_LON_Device1.Command(0).UCPTcommand = New iLON_SmartServer.LON_Device_eCommand()
my_LON_Device1.Command(0).UCPTcommand =
   iLON_SmartServer.LON_Device_eCommand.ChangeCommissionStatus
my_LON_Device1.Command(0).UCPTstatus = New iLON_SmartServer.E_LonString()
my_LON_Device1.Command(0).UCPTstatus.LonFormat = "UCPTstatus"
my_LON_Device1.Command(0).UCPTstatus.Value = "STATUS_REQUEST"

'run device application
my_LON_Device1.Command(1) = New iLON_SmartServer.LON_Device_CfgCommand()
my_LON_Device1.Command(1).UCPTcommand = New iLON_SmartServer.LON_Device_eCommand()
my_LON_Device1.Command(1).UCPTcommand =
   iLON_SmartServer.LON_Device_eCommand.ChangeApplicationStatus
my_LON_Device1.Command(1).UCPTstatus = New iLON_SmartServer.E_LonString()
my_LON_Device1.Command(1).UCPTstatus.LonFormat = "UCPTstatus"
my_LON_Device1.Command(1).UCPTstatus.Value = "STATUS_REQUEST"

'get the device template to show FBs and DPs in Web UI
my_LON_Device1.Command(2) = New iLON_SmartServer.LON_Device_CfgCommand()
my_LON_Device1.Command(2).UCPTcommand = New iLON_SmartServer.LON_Device_eCommand()
my_LON_Device1.Command(2).UCPTcommand = iLON_SmartServer.LON_Device_eCommand.GetTemplate
my_LON_Device1.Command(2).UCPTstatus = New iLON_SmartServer.E_LonString()
my_LON_Device1.Command(2).UCPTstatus.LonFormat = "UCPTstatus"
my_LON_Device1.Command(2).UCPTstatus.Value = "STATUS_REQUEST"

'=====CREATING AND INSTALLING LON DEVICE #2==================

' specify properties of new LON Device #2
my_LON_Device2.UCPTname = "Net/LON/DIO-2"
my_LON_Device2.UCPTlocal = 0
my_LON_Device2.UCPTuniqueId = HexStringToArray("00a145784600")
my_LON_Device2.UCPTprogramId = HexStringToArray("80000105288a0403")
my_LON_Device2.UCPTurlTemplate = "/root/lonWorks/Import/Echelon/LonPoint/Version3/dio-10v3.XIF"
my_LON_Device2.UCPTcommissionStatus = New iLON_SmartServer.E_LonString()
my_LON_Device2.UCPTcommissionStatus.Value = "COMMISSIONED"
my_LON_Device2.UCPTapplicationStatus = New iLON_SmartServer.E_LonString()
my_LON_Device2.UCPTapplicationStatus.Value = "APP_RUNNING"

'create a command array to store device commands to be sent
my_LON_Device2.Command = New iLON_SmartServer.LON_Device_CfgCommand(2) {}

'commission device
my_LON_Device2.Command(0) = New iLON_SmartServer.LON_Device_CfgCommand()
my_LON_Device2.Command(0).UCPTcommand = New iLON_SmartServer.LON_Device_eCommand()
my_LON_Device2.Command(0).UCPTcommand =
   iLON_SmartServer.LON_Device_eCommand.ChangeCommissionStatus
my_LON_Device2.Command(0).UCPTstatus = New iLON_SmartServer.E_LonString()
my_LON_Device2.Command(0).UCPTstatus.LonFormat = "UCPTstatus"
my_LON_Device2.Command(0).UCPTstatus.Value = "STATUS_REQUEST"

'run device application
my_LON_Device2.Command(1) = New iLON_SmartServer.LON_Device_CfgCommand()
my_LON_Device2.Command(1).UCPTcommand = New iLON_SmartServer.LON_Device_eCommand()
my_LON_Device2.Command(1).UCPTcommand =
   iLON_SmartServer.LON_Device_eCommand.ChangeApplicationStatus
my_LON_Device2.Command(1).UCPTstatus = New iLON_SmartServer.E_LonString()
my_LON_Device2.Command(1).UCPTstatus.LonFormat = "UCPTstatus"
my_LON_Device2.Command(1).UCPTstatus.Value = "STATUS_REQUEST"
```

```
        'get the device template to show FBs and DPs in Web UI
        my_LON_Device2.Command(2) = New iLON_SmartServer.LON_Device_CfgCommand()
        my_LON_Device2.Command(2).UCPTcommand = New iLON_SmartServer.LON_Device_eCommand()
        my_LON_Device2.Command(2).UCPTcommand = iLON_SmartServer.LON_Device_eCommand.GetTemplate
        my_LON_Device2.Command(2).UCPTstatus = New iLON_SmartServer.E_LonString()
        my_LON_Device2.Command(2).UCPTstatus.LonFormat = "UCPTstatus"
        my_LON_Device2.Command(2).UCPTstatus.Value = "STATUS_REQUEST"

        'Call the Set() function

        Dim Device_Return_ItemColl As iLON_SmartServer.Item_Coll = SmartServer._iLON.Set(ItemCfgColl)

        Device_Return_ItemColl.xSelect = "//Item[@xsi:type=""LON_Device_Cfg""]"

        If Device_Return_ItemColl.UCPTfaultCount > 0 Then
            ' print out error and exit
            Console.Out.WriteLine("An error occurred:")

            For j As Integer = 0 To Device_Return_ItemColl.Item.Length - 1
                If Device_Return_ItemColl.Item(j).fault IsNot Nothing Then
                    Console.Out.WriteLine(("Item: " & Device_Return_ItemColl.Item(j).UCPTname &
                      ", fault code: " & Device_Return_ItemColl.Item(j).fault.faultcode.Value &
                      ", fault string: ") & Device_Return_ItemColl.Item(j).fault.faultstring)
                End If
            Next
        Else
            ItemCfgColl = SmartServer._iLON.Get(Device_Return_ItemColl)

            For j As Integer = 0 To ItemCfgColl.Item.Length - 1
                Dim newDevice As iLON_SmartServer.LON_Device_Cfg =
                   DirectCast(ItemCfgColl.Item(j), iLON_SmartServer.LON_Device_Cfg)
                Console.WriteLine((("New Device Created = " & newDevice.UCPTname & ".  Status = ") +
                   newDevice.UCPTcommissionStatus.Value & " and ") +
                   newDevice.UCPTapplicationStatus.Value & "." & vbCr)
            Next
        End If

        Console.ReadLine()

      Finally

       SmartServer.CloseBindingToSmartServer()

      End Try
    End Sub
End Module
```

21.2.5 Commissioning External Devices in Visual Basic.NET

This VB console example reads the <UCPTitemStatus> of external LonWorks devices (obtained using an xSelect), commissions any unconfigured devices, and reports the status of the network management commands. Note that this example does not include code for checking that all the network management commands have been completed before terminating.

You can execute this code after you have referenced and inherited from the SmartServer WSDL as described in section 20.1, and instantiated and initialized the Web service client as described in section 20.2.3.

For more information on the network management commands issued in this example, see section 14.3.3.1, *Issuing Network Management Commands*.

```
Module CommDeviceModule

    Sub Main()

        'See Section 20.2.3 for more information on iLON_SoapCalls class
         Dim SmartServer As iLON_SoapCalls = New iLON_SoapCalls
         SmartServer.BindClientToSmartServer()

        Try
```

```
'we create an xSelect object and then specify the filter to be used
 Dim xSelect_Device As ILON_SmartServer.E_xSelect = New iLON_SmartServer.E_xSelect

 xSelect_Device.xSelect =
"//Item[@xsi:type=""LON_Device_Cfg""][UCPTitemStatus=""IS_UNCONFIGURED""]"

 'we create an ItemColl that stores the objects returned by a List() function that takes our
 'xSelect object

 Dim ItemColl_Device As ILON_SmartServer.Item_Coll = SmartServer._iLON.List(xSelect_Device)

 'we use an xSelect to further filter the items returned by the List() function

 ItemColl_Device.xSelect = "//Item[@xsi:type=""LON_Device_Cfg""][UCPTlocal =""0""]"

 'we create an ItemCfgColl that stores the objects returned by a Get() function that takes the
 'ItemColl returned by the List()

 Dim ItemCfgColl As ILON_SmartServer.Item_CfgColl = SmartServer._iLON.Get(ItemColl_Device)

 'check that there are obejcts in the ItemCfgColl

 If (ItemCfgColl.UCPTfaultCount > 0) Then

     Console.Out.WriteLine("you've got Get errors")

 Else

     For i As Integer = 0 To ItemCfgColl.Item.Length - 1

         ' we allocate a Item-Data array object to read Device configurations

         Dim deviceItems As iLON_SmartServer.LON_Device_Cfg = ItemCfgColl.Item(i)
         Console.Out.WriteLine(deviceItems.UCPTname + ", STATUS = " +
         deviceItems.UCPTitemStatus.Value + vbNewLine)

         deviceItems.UCPTcommissionStatus.Value = "COMMISSIONED"
         deviceItems.UCPTapplicationStatus.Value = "APP_RUNNING"

         ReDim deviceItems.Command(0 To 2)

         deviceItems.Command(0) = New iLON_SmartServer.LON_Device_CfgCommand
         deviceItems.Command(0).UCPTcommand =
           iLON_SmartServer.LON_Device_eCommand.ChangeApplicationStatus
         deviceItems.Command(0).UCPTstatus = New iLON_SmartServer.E_LonString
         deviceItems.Command(0).UCPTstatus.Value = "STATUS_REQUEST"

         deviceItems.Command(1) = New iLON_SmartServer.LON_Device_CfgCommand
         deviceItems.Command(1).UCPTcommand =
           iLON_SmartServer.LON_Device_eCommand.ChangeCommissionStatus
         deviceItems.Command(1).UCPTstatus = New iLON_SmartServer.E_LonString
         deviceItems.Command(1).UCPTstatus.Value = "STATUS_REQUEST"

         deviceItems.Command(2) = New iLON_SmartServer.LON_Device_CfgCommand
         deviceItems.Command(2).UCPTcommand = iLON_SmartServer.LON_Device_eCommand.Reset
         deviceItems.Command(2).UCPTstatus = New iLON_SmartServer.E_LonString
         deviceItems.Command(2).UCPTstatus.Value = "STATUS_REQUEST"

         Console.Out.WriteLine(vbNewLine + "*DEVICE CONFIGURATION CHECK*" + vbNewLine +
         vbNewLine + deviceItems.UCPTname + vbNewLine +
        "STATUS = " + deviceItems.UCPTitemStatus.Value + vbNewLine +
        "COMMISSION STATUS = " + deviceItems.UCPTcommissionStatus.Value + vbNewLine +
        "APPLICATION STATUS = " + deviceItems.UCPTapplicationStatus.Value + vbNewLine)

     Next

 End If

 Dim ItemColl_SetReturn As ILON_SmartServer.Item_Coll = SmartServer._iLON.Set(ItemCfgColl)
```

```
        ItemColl_SetReturn.xSelect = "//Item[@xsi:type=""LON_Device_Cfg""]"

        Dim ItemCfgColl_SetReturn As ILON_SmartServer.Item_CfgColl = SmartServer._iLON.Get(ItemColl_SetReturn)

        If (ItemCfgColl_SetReturn.UCPTfaultCount > 0) Then

            Console.Out.WriteLine("you've got Get errors")

        Else

            For i As Integer = 0 To ItemCfgColl_SetReturn.Item.Length - 1

                Console.Out.WriteLine(vbNewLine + "*INSTALLATION STATUS CHECK*" + vbNewLine +
                vbNewLine)

                Dim deviceItemsCheck As iLON_SmartServer.LON_Device_Cfg =
                 ItemCfgColl_SetReturn.Item(i)

                 Console.Out.WriteLine(deviceItemsCheck.UCPTname & vbNewLine +
                "COMMISSION REQUEST STATUS = " + deviceItemsCheck.Command(0).UCPTstatus.Value +
                 vbNewLine +
                "ONLINE REQUEST STATUS = " + deviceItemsCheck.Command(1).UCPTstatus.Value + vbNewLine
                + "RESET REQUEST STATUS = " + deviceItemsCheck.Command(2).UCPTstatus.Value +
                vbNewLine)

                 Do Until ((deviceItemsCheck.Command(0).UCPTstatus.Value = "STATUS_DONE") And
                 (deviceItemsCheck.Command(1).UCPTstatus.Value = "STATUS_DONE") And
                 (deviceItemsCheck.Command(2).UCPTstatus.Value = "STATUS_DONE"))

                    Threading.Thread.Sleep(1500)

                    ItemCfgColl_SetReturn = SmartServer._iLON.Get(ItemColl_SetReturn)

                    For j As Integer = 0 To ItemCfgColl_SetReturn.Item.Length – 1

                        Console.Out.WriteLine(vbNewLine + "*INSTALLATION STATUS CHECK*" + vbNewLine)
                        deviceItemsCheck = ItemCfgColl_SetReturn.Item(j)
                        Console.Out.WriteLine(deviceItemsCheck.UCPTname & vbNewLine +
                        "COMMISSION REQUEST STATUS = " + deviceItemsCheck.Command(0).UCPTstatus.Value
                        + vbNewLine +
                       "ONLINE REQUEST STATUS = " + deviceItemsCheck.Command(1).UCPTstatus.Value +
                        vbNewLine +
                       "RESET REQUEST STATUS = " + deviceItemsCheck.Command(2).UCPTstatus.Value +
                        vbNewLine)
                    Next
                Loop
                Console.Out.WriteLine(vbNewLine + "*DEVICE INSTALLATION COMPLETE*" + vbNewLine +
                vbNewLine + deviceItemsCheck.UCPTname & vbNewLine + vbNewLine +
                "COMMISSION REQUEST STATUS = " + deviceItemsCheck.Command(0).UCPTstatus.Value +
                 vbNewLine +
                "ONLINE REQUEST STATUS = " + deviceItemsCheck.Command(1).UCPTstatus.Value +
                 vbNewLine +
                "RESET REQUEST STATUS = " + deviceItemsCheck.Command(2).UCPTstatus.Value +
                 vbNewLine)
            Next

        End If

        Console.ReadLine()

      Finally

       SmartServer.CloseBindingToSmartServer()

      End Try

  End Sub

End Module
```

21.2.6 Discovering and Installing External Devices in Visual Basic.NET

This console example scans a LONWORKS network for uncommissioned devices, processes the Neuron ID and program ID data of the discovered devices, and then commissions the devices, starts the devices' applications, and gets the devices' templates (to display the devices' functional blocks and data points in the SmartServer Web interface). The example then prints out the names and statuses of the devices that have been installed.

You can execute this code after you have referenced and inherited from the SmartServer WSDL as described in section 20.1, and instantiated and initialized the Web service client as described in section 20.2.3.

For more information on discovering uncommissioned LONWORKS devices, see section 14.1.3.2, *Issuing Network Scan Commands to Discover Devices*. You must also upload the device interface (XIF) files of the devices you are discovering and installing to the root/LonWorks/import folder on the SmartServer flash disk, or create device templates (XML files) for the devices.

```
Imports System.Threading

Module Module1
    'Function required for converting device Neuron IDs and program IDs to a byte[]
    Public Function HexStringToArray(ByVal str As String) As Byte()
        Dim nLen As Integer = str.Length / 2
        Dim arr As Byte() = New Byte(nLen - 1) {}

        For i As Integer = 0 To nLen - 1
            Dim strByte As String = str.Substring(i * 2, 2)
            arr(i) = [Byte].Parse(strByte, System.Globalization.NumberStyles.HexNumber)
        Next
        Return arr
    End Function

    Public Sub Main()

        Dim SmartServer As iLON_SoapCalls = New iLON_SoapCalls
        SmartServer.BindClientToSmartServer()

        Try
            'create LONNetworkScanCommandInvoke item and ScanCommand attribute
            Dim networkScan As New iLON_SmartServer.LON_Network_ScanCommand_Invoke()
            networkScan.ScanCommand = iLON_SmartServer.LON_Network_eScanCommand.SetScan

            '***set LONNetworkScanCommandInvoke properties***

            '1. Set network UCPTname
            networkScan.UCPTname = "Net"

            '2. Set Scan Command

            'a. set scan frequency
            Dim scanFrequency As New iLON_SmartServer.LON_Network_ScanCommand_InvokeCommand()
            scanFrequency.UCPTcommand = iLON_SmartServer.LON_Device_IlonNi_eCommand.ScanOnce

            'b. set scan status
            Dim scanStatus As New iLON_SmartServer.E_LonString()
            scanStatus.LonFormat = "UCPTstatus"
            scanStatus.Value = "STATUS_REQUEST"
            scanFrequency.UCPTstatus = scanStatus

            'c. add scan command to LONNetworkScanCommandInvoke item
            networkScan.Command = New iLON_SmartServer.LON_Network_ScanCommand_InvokeCommand(0) {}
            networkScan.Command(0) = scanFrequency

            '3. Set UCPTscan
            Dim domain As New iLON_SmartServer.E_LonString()
            domain.LonFormat = "ucptScan"
            domain.Value = "NST_ILON_DOMAIN"
            networkScan.UCPTscan = New iLON_SmartServer.E_LonString(0) {}
```

```
networkScan.UCPTscan(0) = domain

'send InvokeCmd

Dim itemColl As New iLON_SmartServer.Item_Coll()
itemColl.Item = New iLON_SmartServer.Item(0) {}
itemColl.Item(0) = networkScan
SmartServer._iLON.InvokeCmd(itemColl)

Console.WriteLine("starting scan")

'send the GetScan command to check network scan progress
Dim networkScan_Check As New iLON_SmartServer.LON_Network_ScanCommand_Invoke()
networkScan_Check.ScanCommand = iLON_SmartServer.LON_Network_eScanCommand.GetScan
networkScan_Check.UCPTname = "Net"

Dim itemColl_Check As New iLON_SmartServer.Item_Coll()
itemColl_Check.Item = New iLON_SmartServer.Item(0) {}
itemColl_Check.Item(0) = networkScan_Check

'Check scan status
Dim scanDone As Boolean = False
While Not scanDone
    SmartServer._iLON.InvokeCmd(itemColl_Check)

    Dim scanCheck_Response As New iLON_SmartServer.InvokeCmdResponse()
    scanCheck_Response.iLonItem = itemColl_Check
    Dim scanStatusCheck As iLON_SmartServer.LON_Network_ScanCommand_Invoke =
        DirectCast(scanCheck_Response.iLonItem.Item(0),
        iLON_SmartServer.LON_Network_ScanCommand_Invoke)

    'if the scan is done set scanDone flag to true
    If scanStatusCheck.Command(0).UCPTstatus.Value = "STATUS_DONE" Then
        Console.WriteLine("Network Scan Status = " &
          scanStatusCheck.Command(0).UCPTstatus.Value)
        scanDone = True

   'if the scan is not done, keep scanDone flag at false, wait 10 seconds, and check again
    ElseIf scanStatusCheck.Command(0).UCPTstatus.Value = "STATUS_PENDING" Then
        Console.WriteLine("Network Scan Status = " &
          scanStatusCheck.Command(0).UCPTstatus.Value)
        Thread.Sleep(10000)
    End If
End While
' A "<network>/#DeviceDiscovery" data logger is automatically created by the network scan
' read the Data Logger and process the data of the discovered data
Dim deviceDiscovered As New iLON_SmartServer.UFPTdataLogger_Data()
deviceDiscovered.UCPTname = "Net/#DeviceDiscovery"
Dim itemColl_DataLog As New iLON_SmartServer.Item_Coll()
itemColl_DataLog.xSelect = "//Item[@xsi:type=""UFPTdataLogger_Data""]"
itemColl_DataLog.Item = New iLON_SmartServer.Item(0) {}
itemColl_DataLog.Item(0) = deviceDiscovered

Dim dataLogger As iLON_SmartServer.Item_DataColl =
 SmartServer._iLON.Read(itemColl_DataLog)
Console.WriteLine("Devices Discovered = " & (dataLogger.Item.Length - 1))
Console.WriteLine("=====================================================")

Dim itemCfgColl As New iLON_SmartServer.Item_CfgColl()

'Create a new ItemCfgColl to store discovered devices
itemCfgColl.Item = New iLON_SmartServer.Item_Cfg(dataLogger.Item.Length - 2) {}

For i As Integer = 1 To dataLogger.Item.Length - 1
    'we start at 1 to account for the metaData item in Data Logger
    Dim dataLoggerData As iLON_SmartServer.UFPTdataLogger_Data =
      DirectCast(dataLogger.Item(i), iLON_SmartServer.UFPTdataLogger_Data)

    If dataLoggerData IsNot Nothing Then
        Console.WriteLine(("Device #" & i & ": Neuron ID and Program ID = ") &
         dataLoggerData.UCPTvalue(0).Value)
```

```
End If
' -------------- CREATING DISCOVERED LONWORKS DEVICES--------------

'Create a new LON_Device_Cfg Item and add it to ItemCfgColl
Dim my_LON_Device As New iLON_SmartServer.LON_Device_Cfg()
itemCfgColl.Item(i - 1) = my_LON_Device
'subtract 1 for the metaData item in Data Logger
'parse Neuron ID and Program ID from Data Logger
Dim NID_PID As [String] = dataLoggerData.UCPTvalue(0).Value
Dim NID As [String] = NID_PID.Substring(0, 12)
Console.WriteLine("Neuron ID = " & NID)
Dim PID As [String] = NID_PID.Substring(13, 16)
Console.WriteLine("Program ID = " & PID)

'set Neuron ID, which is a byte[]
my_LON_Device.UCPTuniqueId = (HexStringToArray(NID))

'set Program ID, which is a byte[]
my_LON_Device.UCPTprogramId = (HexStringToArray(PID))

'set template
Dim xSelect As New iLON_SmartServer.E_xSelect()
xSelect.xSelect = "//Item[@xsi:type=""TemplateManager_Cfg""]
                  [UCPTfileType=""TEMPLATE_OR_XIF""]
                  [UCPTprogramId=""" & PID & """]"
itemColl = SmartServer._iLON.List(xSelect)

Dim template As iLON_SmartServer.TemplateManager_Surrogate_Cfg =
   DirectCast(itemColl.Item(0), iLON_SmartServer.TemplateManager_Surrogate_Cfg)
Dim templateName As [String] = template.UCPTname
Console.WriteLine("Device Template = " & templateName)
my_LON_Device.UCPTurlTemplate = templateName

'set the device name

'1. get the name of the channel ("Net/LON")
xSelect.xSelect = "//Item[@xsi:type=""LON_Channel_Cfg""][UCPThidden=0]"
itemColl = SmartServer._iLON.List(xSelect)
Dim channel As iLON_SmartServer.Item = itemColl.Item(0)

'2. get the name of the xif
Dim separator As String() = New String() {"/"}
Dim templateName_justxif As [String]() = templateName.Split(separator, 0)
Dim templateNameLength As Integer = templateName_justxif.Length
Dim xifName As [String] = templateName_justxif(templateNameLength - 1)
Console.WriteLine("XIF Name = " & xifName)

'3. name device using channel name, /device [index], and xif name ("Net/LON/Device 1 (ai-10v3.xif)")
Dim deviceName As [String] = ((channel.UCPTname & "/" & "Device ") & i & " (") &
  xifName & ")"
Console.WriteLine("Device Name = " & deviceName)
Console.WriteLine("=======================================================")
my_LON_Device.UCPTname = deviceName

'set Commission status
Dim commissionStatus_LonString As New iLON_SmartServer.E_LonString()
commissionStatus_LonString.Value = "COMMISSIONED"
my_LON_Device.UCPTcommissionStatus = commissionStatus_LonString

'set Application status
Dim applicationStatus_LonString As New iLON_SmartServer.E_LonString()
applicationStatus_LonString.Value = "APP_RUNNING"
my_LON_Device.UCPTapplicationStatus = applicationStatus_LonString

'set tree and app icon; based on program ID
my_LON_Device.UCPTannotation = PID

'&&&&send device commands&&&&&&

'commission device
'create a command array to store device commands to be sent
```

```
            my_LON_Device.Command = New iLON_SmartServer.LON_Device_CfgCommand(2) {}

            'commission device
            my_LON_Device.Command(0) = New iLON_SmartServer.LON_Device_CfgCommand()
            my_LON_Device.Command(0).UCPTcommand = New iLON_SmartServer.LON_Device_eCommand()
            my_LON_Device.Command(0).UCPTcommand =
             iLON_SmartServer.LON_Device_eCommand.ChangeCommissionStatus
            my_LON_Device.Command(0).UCPTstatus = New iLON_SmartServer.E_LonString()
            my_LON_Device.Command(0).UCPTstatus.LonFormat = "UCPTstatus"
            my_LON_Device.Command(0).UCPTstatus.Value = "STATUS_REQUEST"

            'run device application
            my_LON_Device.Command(1) = New iLON_SmartServer.LON_Device_CfgCommand()
            my_LON_Device.Command(1).UCPTcommand = New iLON_SmartServer.LON_Device_eCommand()
            my_LON_Device.Command(1).UCPTcommand =
              iLON_SmartServer.LON_Device_eCommand.ChangeApplicationStatus
            my_LON_Device.Command(1).UCPTstatus = New iLON_SmartServer.E_LonString()
            my_LON_Device.Command(1).UCPTstatus.LonFormat = "UCPTstatus"
            my_LON_Device.Command(1).UCPTstatus.Value = "STATUS_REQUEST"

            'get the device template to show FBs and DPs in Web UI
            my_LON_Device.Command(2) = New iLON_SmartServer.LON_Device_CfgCommand()
            my_LON_Device.Command(2).UCPTcommand = New iLON_SmartServer.LON_Device_eCommand()
            my_LON_Device.Command(2).UCPTcommand =
              iLON_SmartServer.LON_Device_eCommand.GetTemplate
            my_LON_Device.Command(2).UCPTstatus = New iLON_SmartServer.E_LonString()
            my_LON_Device.Command(2).UCPTstatus.LonFormat = "UCPTstatus"
            my_LON_Device.Command(2).UCPTstatus.Value = "STATUS_REQUEST"
        Next

        'Call the Set() function

        Dim Device_Return_ItemColl As iLON_SmartServer.Item_Coll =
         SmartServer._iLON.[Set](itemCfgColl)

        Device_Return_ItemColl.xSelect = "//Item[@xsi:type=""LON_Device_Cfg""]"

        If Device_Return_ItemColl.UCPTfaultCount > 0 Then
            ' print out error and exit
            Console.WriteLine("An error occurred:")

            For j As Integer = 0 To Device_Return_ItemColl.Item.Length - 1
                If Device_Return_ItemColl.Item(j).fault IsNot Nothing Then
                    Console.WriteLine(("Item: " & Device_Return_ItemColl.Item(j).UCPTname
                     & ", fault code: ") & Device_Return_ItemColl.Item(j).fault.faultcode.Value
                     & ", fault string: ") & Device_Return_ItemColl.Item(j).fault.faultstring)
                End If
            Next
        Else
            itemCfgColl = SmartServer._iLON.[Get](Device_Return_ItemColl)

            For j As Integer = 0 To itemCfgColl.Item.Length - 1
                If itemCfgColl.Item(j).fault IsNot Nothing Then
                    Console.WriteLine(("Item: " & itemCfgColl.Item(j).UCPTname &
                     ", fault code: ") & itemCfgColl.Item(j).fault.faultcode.Value &
                     ", fault string: ") & itemCfgColl.Item(j).fault.faultstring)
                Else
                    Dim newDevice As iLON_SmartServer.LON_Device_Cfg =
                      DirectCast(itemCfgColl.Item(j), iLON_SmartServer.LON_Device_Cfg)
                    Console.WriteLine(("New Device Created = " & newDevice.UCPTname &
                       ". Status = ") & newDevice.UCPTcommissionStatus.Value & " and ") &
                       newDevice.UCPTapplicationStatus.Value & ".")
                End If
            Next
        End If
        Console.ReadLine()
    Finally
        SmartServer.CloseBindingToSmartServer()
    End Try
  End Sub
End Module
```

21.2.7 Configuring the SmartServer in Visual Basic.NET

This console example uses the system information methods in the SmartServer's system WSDL (**iLON100_System.wsdl**) to check the SmartServer's current time and system information and then sets a new time. Note that the **iLON_SoapCalls** class references the **iLON100_System** Web service instead of the **iLON100** Web service.

You can execute this code after you have referenced and inherited from the SmartServer WSDL as described in section 20.1.

For more information on the system information properties set in this example, see section 19.1, *System Service Methods*.

Main Program

```
Imports System.Threading 'make sure you add this statement

Module SystemTimeModule

    Sub Main()

        'See Section 20.2.3 for more information on iLON_SoapCalls class
         Dim SmartServer As iLON_SoapCalls = New iLON_SoapCalls
         SmartServer.BindClientToSmartServer()

        Try

         'This code checks the SmartServer's time and system info and then sets a new time

        '------------Checking System Time-----------------------------------------

        Console.Out.WriteLine("Checking the SmartServer's System Time" + vbNewLine)

         Dim time As New iLON_SmartServer_System.messageProperties_system()

         Dim timeData As String =
         "<iLONSystemService><UCPTsystemInfoType>SI_TIME</UCPTsystemInfoType></iLONSystemService>"

         Dim timeResult As String = SmartServer._iLON.SystemService_Read_Info(time, timeData)
         Console.Out.WriteLine(timeResult)
         '------------Checking System Information-----------------------------------------

        Console.Out.WriteLine(vbNewLine + "Checking the SmartServer's System Information" +
         vbNewLine)

         Dim systemInfo As New iLON_SmartServer_System.messageProperties_system()

         Dim staticData As String =
         "<iLONSystemService><UCPTsystemInfoType>SI_STATIC</UCPTsystemInfoType></iLONSystemService>"

         Dim staticResult As String =
         SmartServer._iLON.SystemService_Read_Info(systemInfo, staticData)

         Console.Out.WriteLine(staticResult)

        '------------Changing System Time-----------------------------------------

        Console.Out.WriteLine(vbNewLine + "Changing the SmartServer's System Time" + vbNewLine)

         Dim revisedTime As New iLON_SmartServer_System.messageProperties_system()

         Dim revisedTimeData As String = "<iLONSystemService><TIME>SI_TIME<UCPTsystemTime>
             2008-07-05T10:20:00</UCPTsystemTime></TIME></iLONSystemService>"

         Dim revisedTimeResult As String =
          SmartServer._iLON.SystemService_Write_Info(revisedTime, revisedTimeData)

         Console.Out.WriteLine(revisedTimeResult)

         Console.Out.WriteLine(vbNewLine + "Take a 10-second break to see if time updates properly " +
```

```
        vbNewLine)

        Thread.Sleep(10000)

        Dim newTime As New iLON_SmartServer_System.messageProperties_system()

        Dim newTimeData As String =
        "<iLONSystemService><UCPTsystemInfoType>SI_TIME</UCPTsystemInfoType></iLONSystemService>"

        Dim newTimeResult As String =
        SmartServer._iLON.SystemService_Read_Info(newTime, newTimeData)

        Console.Out.WriteLine(newTimeResult)

        Console.ReadLine()

      Finally

       SmartServer.CloseBindingToSmartServer()

      End Try

    End Sub

End Module
```

Web Service Instantiation in iLON_SoapCalls Class

```
Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text
Imports System.ServiceModel

Public Class iLON_SoapCalls
    'your SmartServer's IpAddress
    Public _iLonEndpointIpAddress As String = "<SmartServer IP address>"

    'your SmartServer's Web service reference
    Public _iLON As iLON_SmartServer_System.iLON100portTypeClient = Nothing

    ''' <summary>
    '''   Instantiates the i.LON web service for
    '''   .NET 3.5
    ''' </summary>
    Public Sub BindClientToSmartServer()
        ' Specify the binding to be used for the client.
        Dim binding As BasicHttpBinding = New BasicHttpBinding()

        ' Initialize the namespace
        binding.Namespace = "http:'wsdl.echelon.com/web_services_ns/ilon100/v4.0/message/"

        ' Obtain the URL of the Web service on the SmartServer.
        Dim endpointAddress As System.ServiceModel.EndpointAddress =
           New System.ServiceModel.EndpointAddress("http://" + _iLonEndpointIpAddress + "/WSDL/iLON100_System.wsdl")

        ' Instantiate the SmartServer Web service object with this address and binding.
        _iLON = New iLON_SmartServer_System.iLON100portTypeClient(binding, endpointAddress)

        ' uncomment the lines below to enable authentication
        ' binding.Security.Mode = System.ServiceModel.BasicHttpSecurityMode.TransportCredentialOnly
        ' binding.Security.Transport.ClientCredentialType =
        ' System.ServiceModel.HttpClientCredentialType.Basic
        ' _iLON.ChannelFactory.Credentials.UserName.UserName = "ilon"
        ' _iLON.ChannelFactory.Credentials.UserName.Password = "ilon"

    End Sub

    ''' <summary>
    '''   Close the i.LON web service
    ''' </summary>

    Public Sub CloseBindingToSmartServer()
```

```
        ' Closing the client gracefully
        ' closes the connection and cleans up resources
        Try
            _iLON.Close()
        Finally
            _iLON = Nothing
        End Try

    End Sub

End Class
```

22 Programming the SmartServer with Java

You can write custom applications for the SmartServer in Java. The SmartServer supports the Java API for XML Web Services (JAX-WS 2.0 and 2.1), which is Java programming language API for creating web services.

22.1 Setting up the Java Programming Environment

To setup your Java programming environment for the SmartServer, you need to download and install the following software:

- Echelon SmartServer JAX-ES programming example.
- Eclipse IDE for Java EE Developers 3.5.
- JAVA Development Kit.
- Maven 2.2.1.

After you download and install this software, you need to add environment variables for the JDK and Maven 2.2.1.

22.1.1 Installing Echelon SmartServer JAX-ES Programming Example

To download and setup the Echelon SmartServer JAX-ES programming example, follow these steps:

1. Download the Echelon SmartServer JAX-ES programming example (.zip file) from the Echelon Web site at *www.echelon.com/downloads*.
2. Extract the .zip file to your computer's local drive (for example, **C:**).
3. Browse to the **C:\eclipse\eclipse 3.5\ilon.ws.clients** folder on your computer and confirm that there is a **jax-ws** folder.

22.1.2 Installing Eclipse IDE for Java EE Developers

To download and setup the Eclipse IDE for JAVA EE Developers, follow these steps:

1. Browse to the Eclipse downloads Web page at *www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/galileo/R/eclipse-jee-galileo-win32.zip*.
2. Download the .zip file.
3. Browse to the **C:\eclipse\eclipse 3.5** folder, and then create a new folder named **eclipse-jee-galileo-win32**.
4. Extract the .zip file to the **C:\eclipse\eclipse 3.5\eclipse-jee-galileo-win32** folder.

22.1.3 Installing the Java Development Kit

To download and setup JAVA Development Kit (JDK), follow these steps:

Browse to the Sun Developer Network (SDN) Java SE download s Web page at *www.java.sun.com/javase/downloads/index.jsp*.

1. Download the **JDK 6 Update 17 with Java EE** bundle.
2. Install the JDK into the **C:\Sun** directory following the setup instructions.
3. Open a command prompt, change the directory to **C:\Sun** (enter **cd c:\Sun**), and then enter the **set** command. This displays your system environment.

22.1.4 Installing Maven 2.2.1

To download and setup Maven 2.2.1, follow these steps:

1. Browse to the Maven download Web page at *http://maven.apache.org/download.html*.

2. Download the Maven 2.2.1 .zip file.
3. Browse to the **C:\eclipse\eclipse 3.5\ilon.ws.clients** folder, and then create a new folder named **tools**.
4. Extract the .zip file to the **C:\eclipse\eclipse 3.5\ilon.ws.clients\tools** folder.

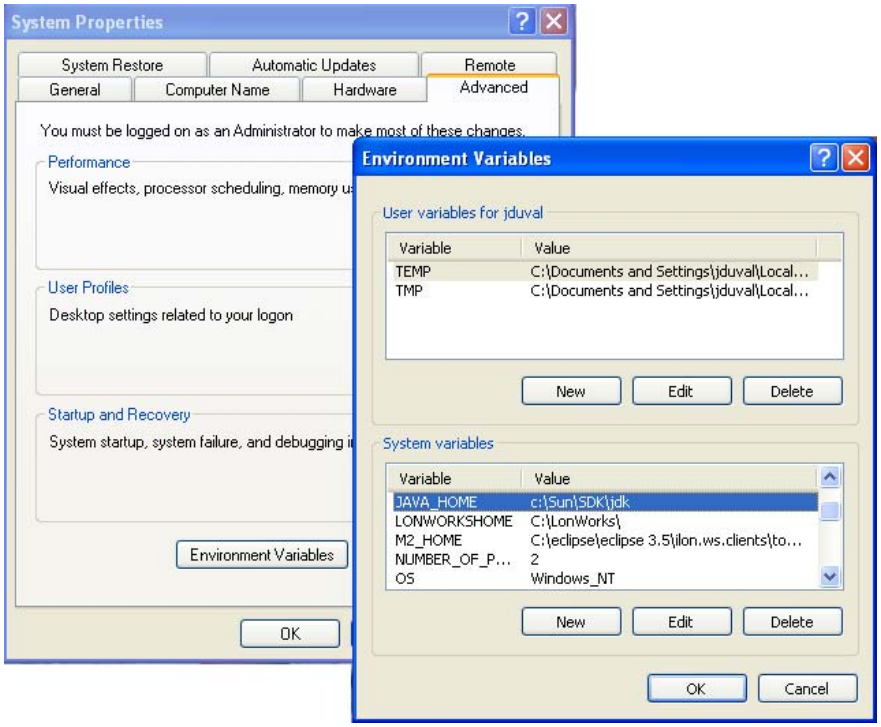
22.1.5 Setting System Environment Variables

To add environment variables for the JDK and Maven 2.2.1, follow these steps:

1. Open the **System Properties** dialog for your computer. To access this dialog, open the Windows Control Panel, and then click **System**. Alternatively, you can right-click the **MyComputer** or **Computer** icon on you desktop and click **Properties**.
2. In the **System Properties** dialog, click **Advanced System Settings** or **Advanced**, and then click the **Environment Variables** tab.
3. Create the system environment variables for the JDK following these steps:
   a. Under **System Variables**, click **New**, enter **JAVA_HOME** in the **Variable Name** property, enter **C:\Sun\SDK\jdk** in the **Variable Value** property, and then click **OK**.
   b. Under **System variables**, click **Path**, and then click **Edit**. Prepend the following variables to the **Variable Value** property (these entries should be listed first):

      ;%JAVA_HOME%\jre\bin;C:\Sun\SDK\bin;

      Note: Delete any other JAVA entries if they are listed (for example, jre-).



4. Create the system environment variables for Maven 2.2.1 following these steps:
   a. Under **System variables**, click **New**, enter **M2_HOME** in the **Variable Name** property, enter **C:\eclipse\eclipse 3.5\ilon.ws.clients\tools\apache-maven-2.2.1** in the **Variable Value** property, and then click **OK**.
   b. Under **System Variables**, click **Path**, and then click **Edit**. Append the following variables to the **Variable Value** property:

      ;%M2_HOME%\bin;%MAVEN_PATH%\bin;

22.2 Creating a JAX-WS Client

To create a JAX-WS client, follow these steps:

1. Create the java proxy classes for the SmartServer's WSDL and create a sample Eclipse project for programming the SmartServer in Java. To do this, follow these steps:

   a. Browse to the **C:\eclipse\eclipse 3.5\ilon.ws.clients\jax-ws\src\wsdl** folder, open the **iLON100.wsdl** file with a text editor, change the IP address at the bottom of the file to the IP address of your SmartServer, and then save the file.

   b. Open a Windows command prompt and change the directory to C:\eclipse\eclipse 3.5\ilon.ws.clients\jax-ws (enter **cd C:\eclipse\eclipse 3.5\ilon.ws.clients\jax-ws**).

   c. This directory contains a **pom.xml** configuration file for Maven that enables you to quickly create a JAX-WS project for a SmartServer client.

   d. Enter the following command to verify that you set the Path system variables correctly:

      mvn –version

      The current version will be displayed if you set the Path system variables correctly.

   e. Enter the following command to download all the required dependencies to your computer and generate the corresponding java proxy classes for the **iLON100.wsdl**. Ignore any warnings that appear.

      mvn clean jaxws:wsimport

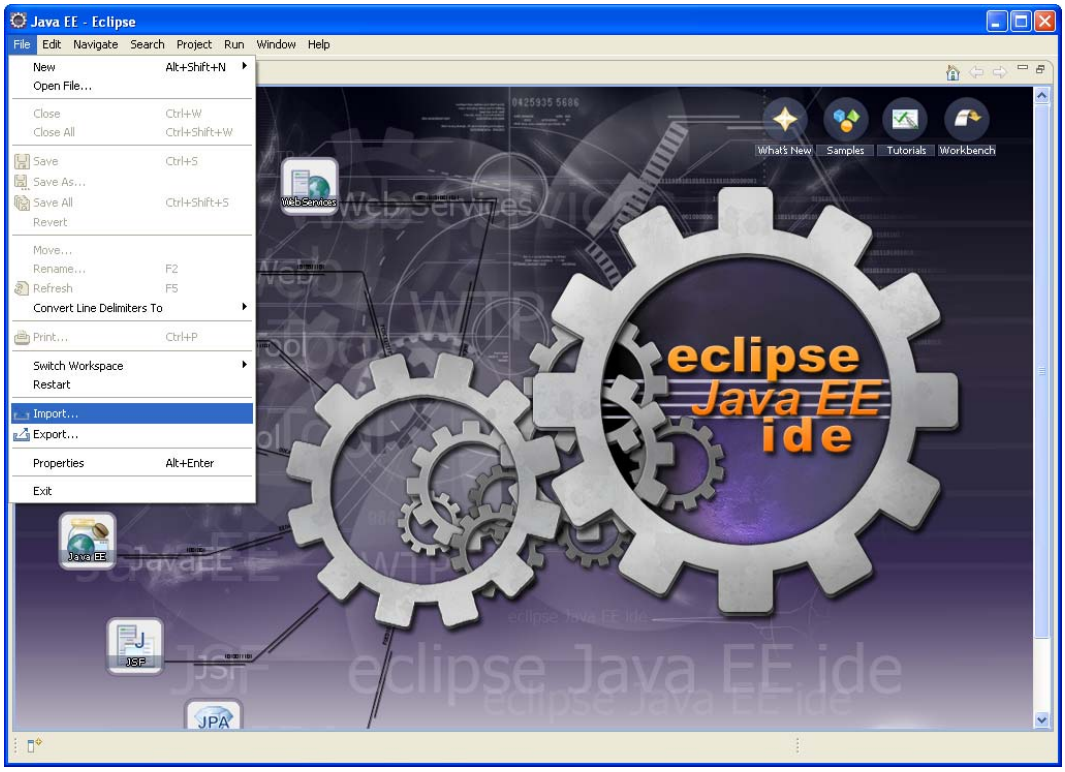
   f. Enter the following command to create an Eclipse project:

      mvn eclipse:eclipse

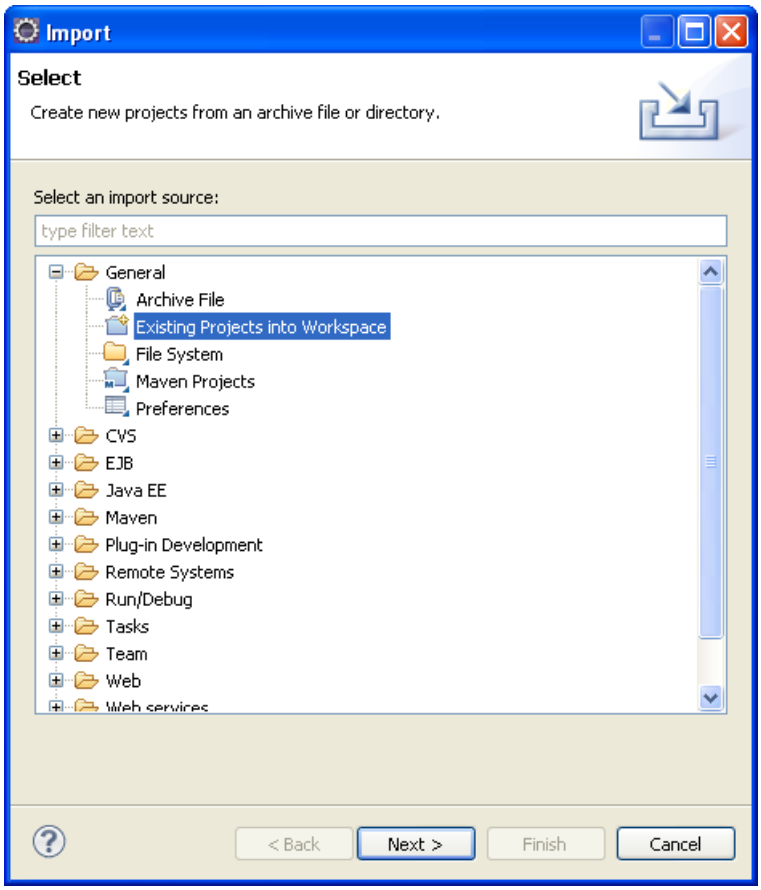
2. Start **Eclipse**. To do this, browse to the **C:\eclipse\eclipse 3.5\eclipse-jee-galieo-win32** folder and then double-click the **eclipse.exe** executable. The Eclipse JAVA EE development tool opens.

3. Import the sample Eclipse project for programming the SmartServer in Java following these steps:

   a. Click **File**, and then click **Import**.



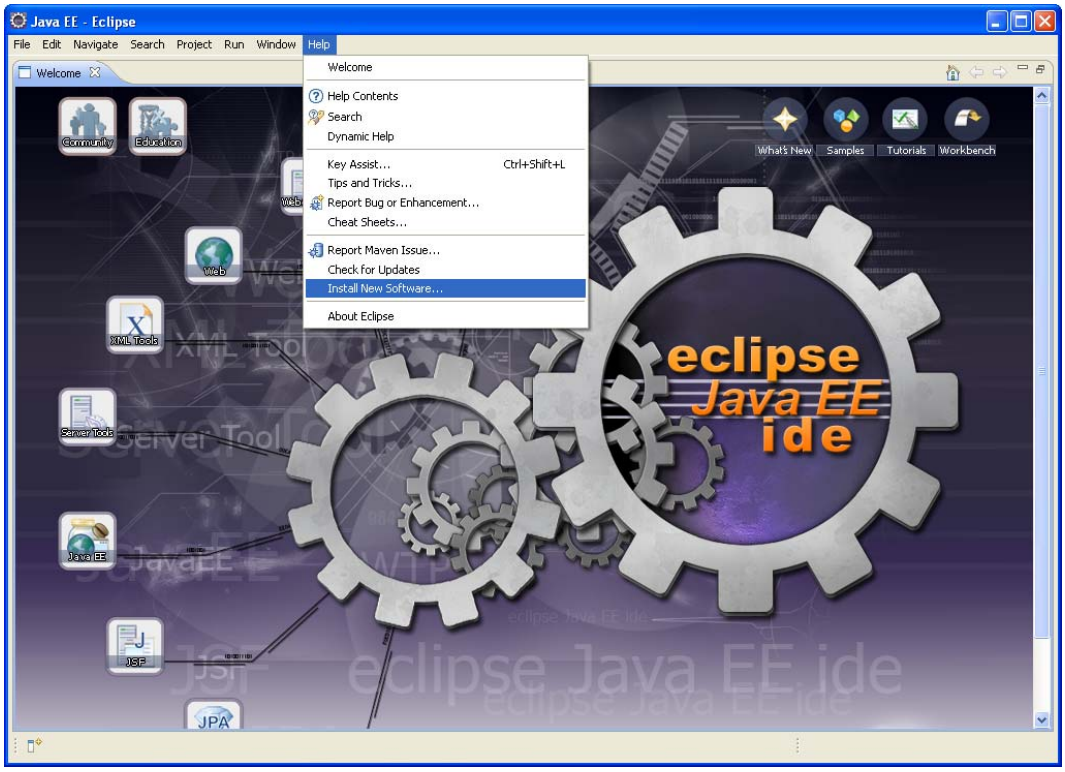
b. The **Import** dialog opens with the Select window. Expand **General**, click **Existing Projects into Workspace**, and then click **Next**.



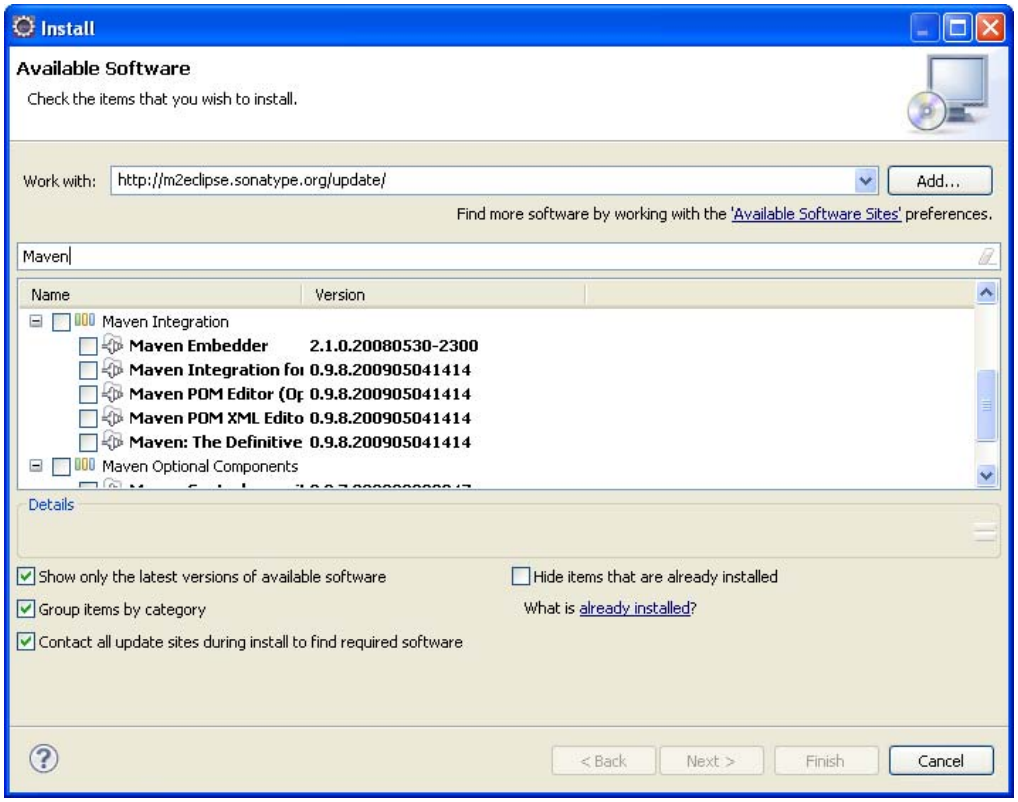
c. The Import window opens. In the **Root Directory** property, enter **C:\eclipse\eclipse 3.5\ilon.ws.clients\jax-ws** and then press ENTER or TAB, or click **Browse** and select the **C:\eclipse\eclipse 3.5\ilon.ws.clients\jax-ws** folder. Click **Finish**.

4. Install Maven integration software following these steps:

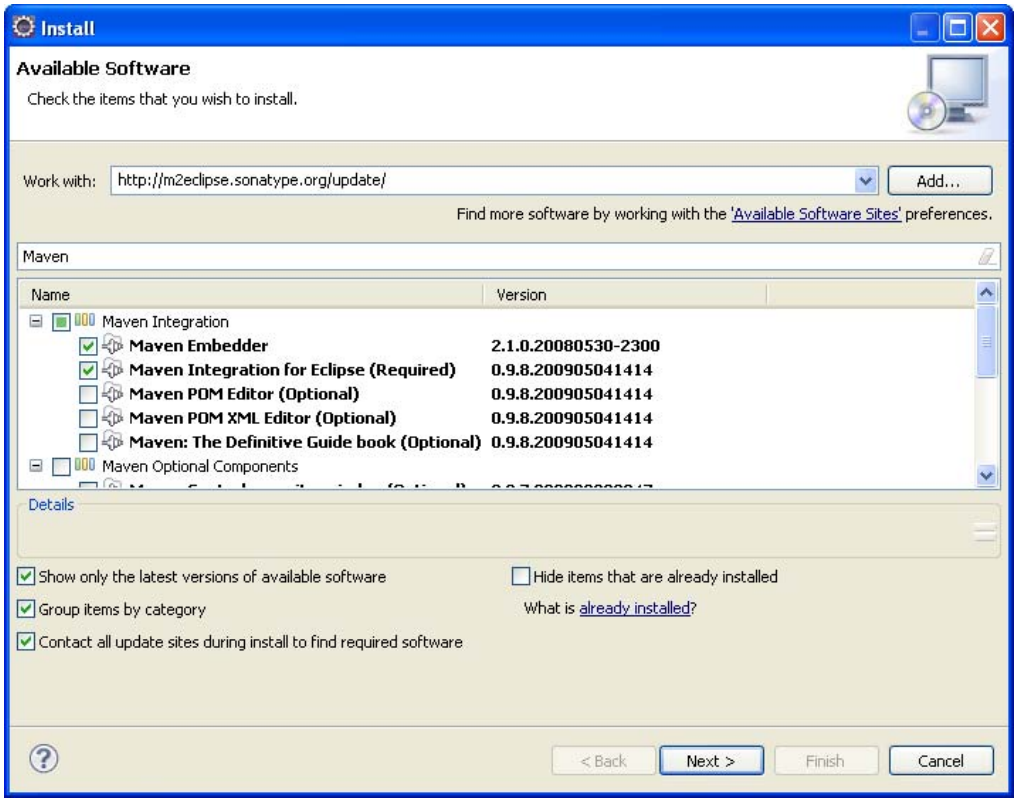
   a. Click **Help** and then click **Install New Software**.



b. In the **Work With** property, enter *http://m2eclipse.sonatype.org/update/*. In the **Type Filter Text property**, enter **Maven**.



c. Under **Maven Integration**, select the **Maven Embedder** and **Maven Integration for Eclipse** check boxes, click **Next**, and then click **Finish**.

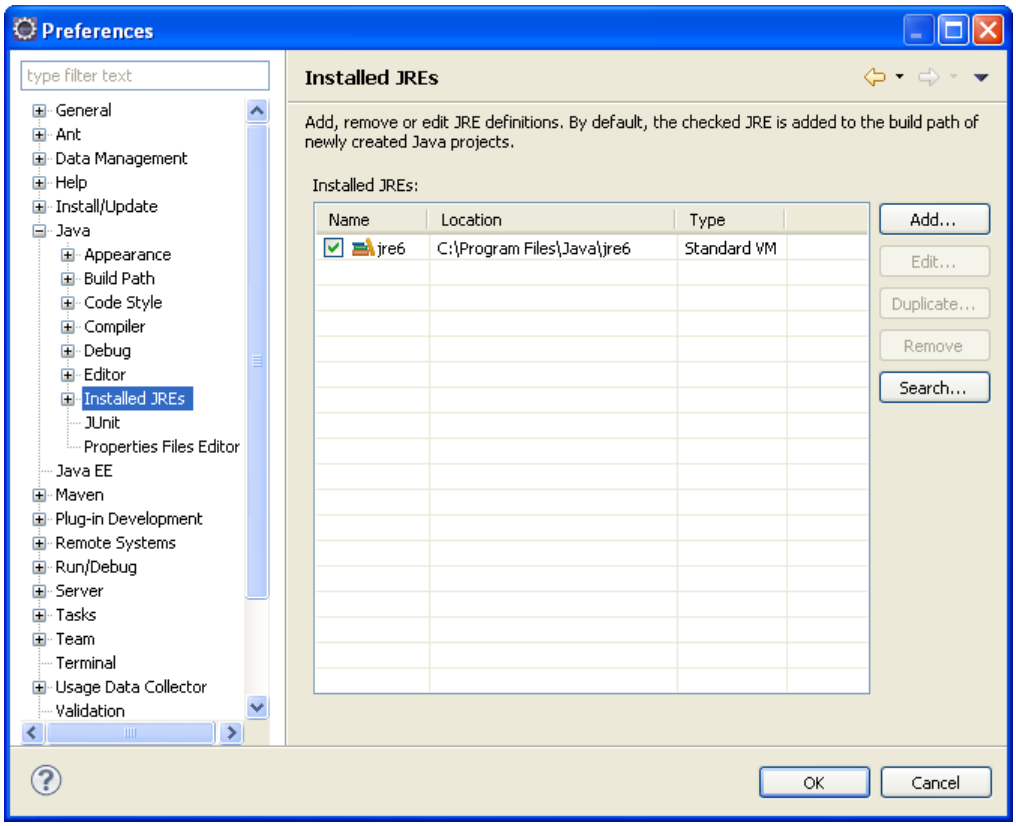


d. Restart Eclipse. To do this, click **File** and then click **Restart**. If the following warning dialog opens, click **OK**.

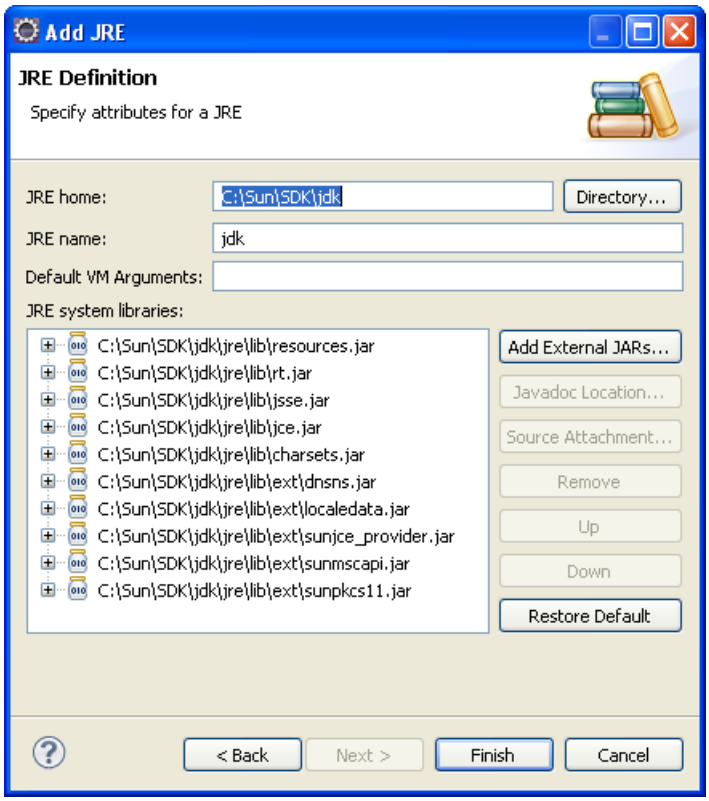
5. Set Eclipse to the JDK you installed in the *Installing the Java Development Kit* section following these steps:

   a. Click **Windows** and then click **Preferences**.

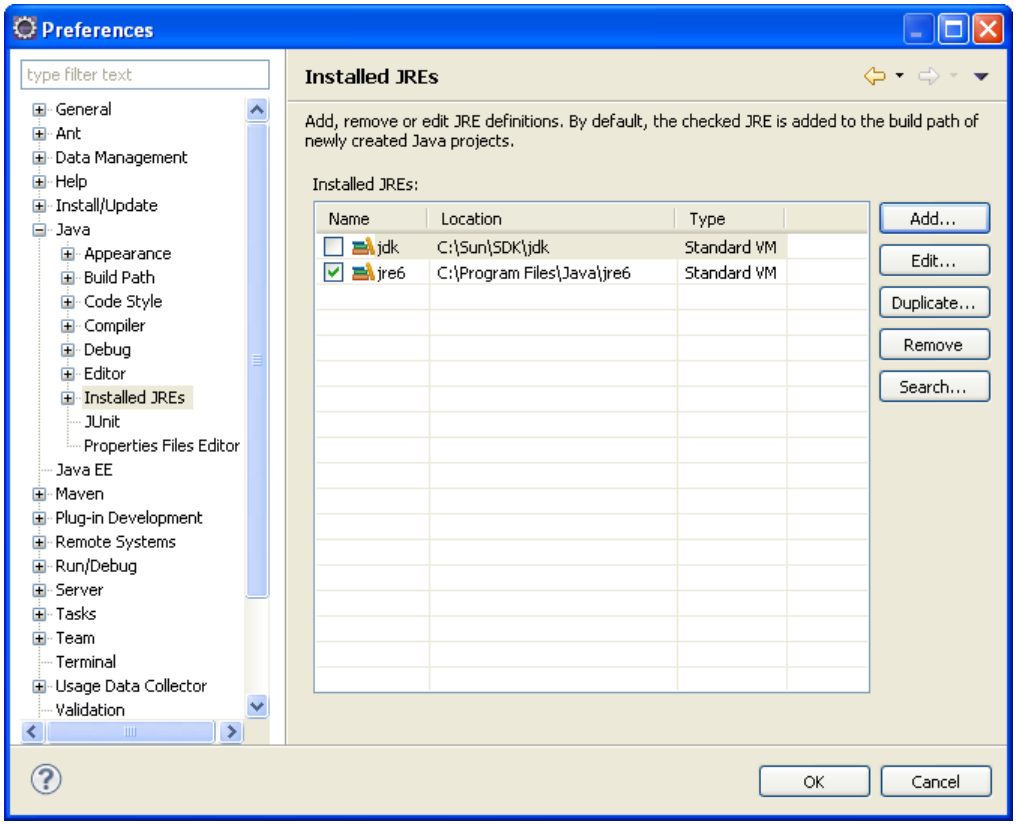
b. The **Preferences** dialog opens. Expand **JAVA** and then click **Installed JREs**.



c. Add the JDK that you installed in the *Installing the Java Development Kit* section. To do this click **Add**, click **Next**, enter **C:\Sun\SDK\jdk** in the **JRE Home** property or click **Directory** and browse to the **C:\Sun\SDK\jdk** folder, and then click **Finish**.



d. A **jdk** entry is listed under **Installed JREs**.



e. Delete the existing **jre6** entry. To do this, click **jre6** and then click **Remove**.

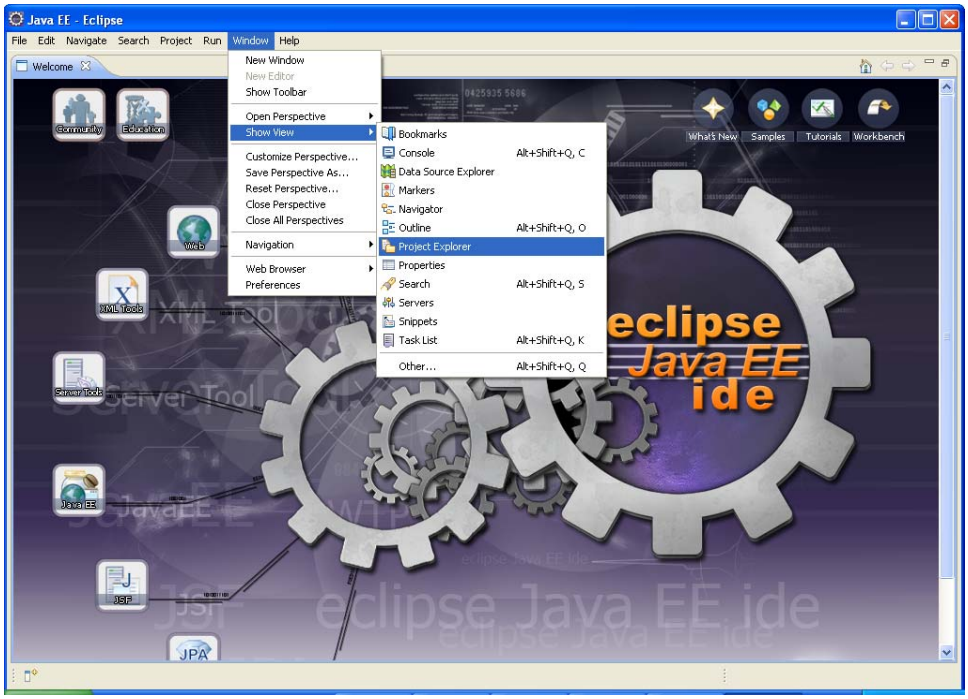
f. Click **OK**.

g. Browse to **C:\WINDOWS\system32** and rename the **java.exe**, **javaw.exe**, **javaws.exe** files to **java.exe_bak**, **javaw.exe_bak**, **javaws.exe_bak**, respectively. This prevents Eclipse from being confused by these files.

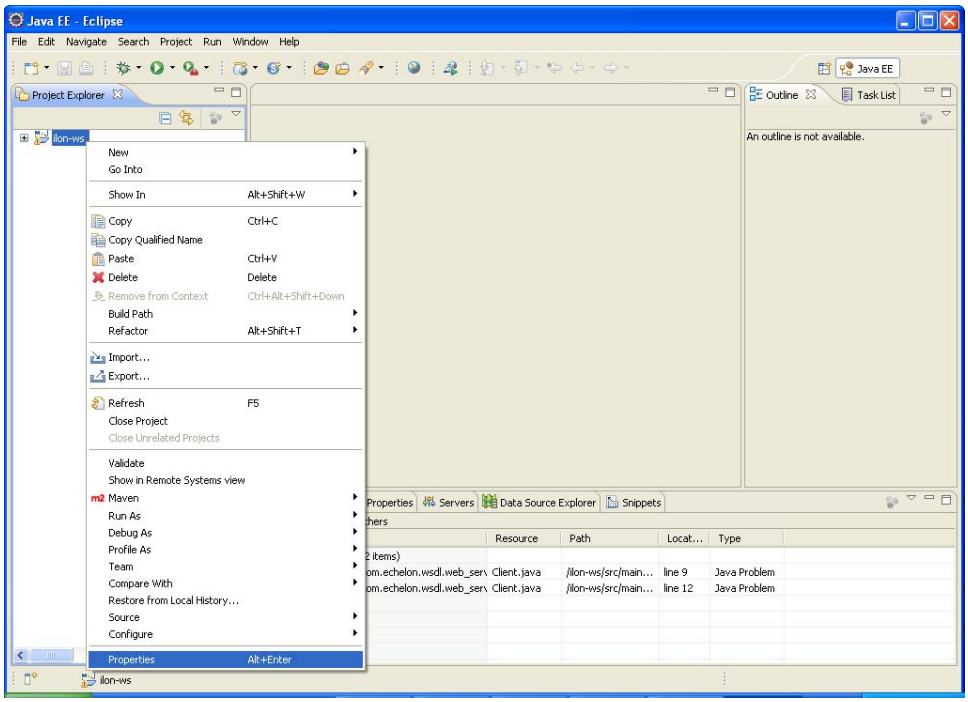
h. Restart Eclipse.

i. Set the Java Build Path to the JDK you installed following these steps:

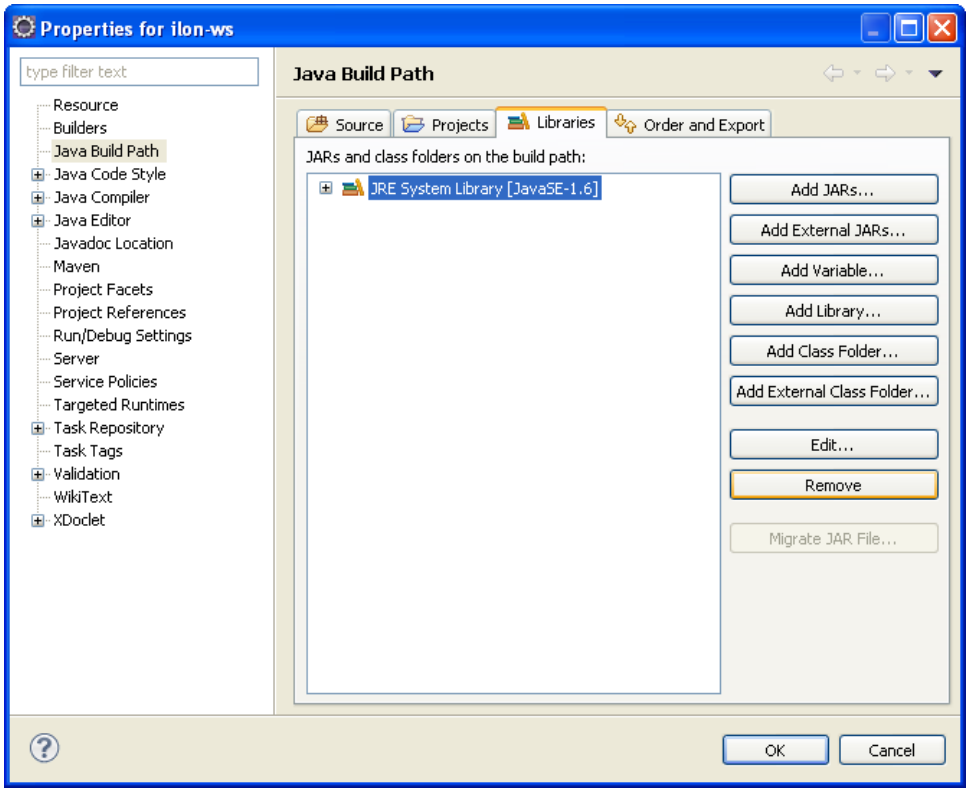
- Open the **Project Explorer** view. To do this, click **Window**, point to **Show View**, and then click **Project Explorer**.



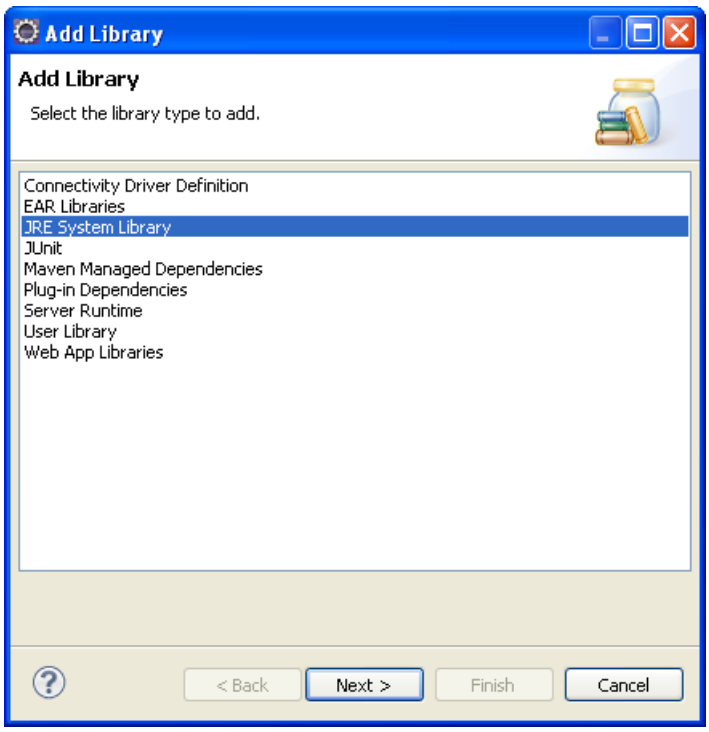
- In the **Project Explorer** view, right-click the **ilon-ws** folder and then click **Properties**.



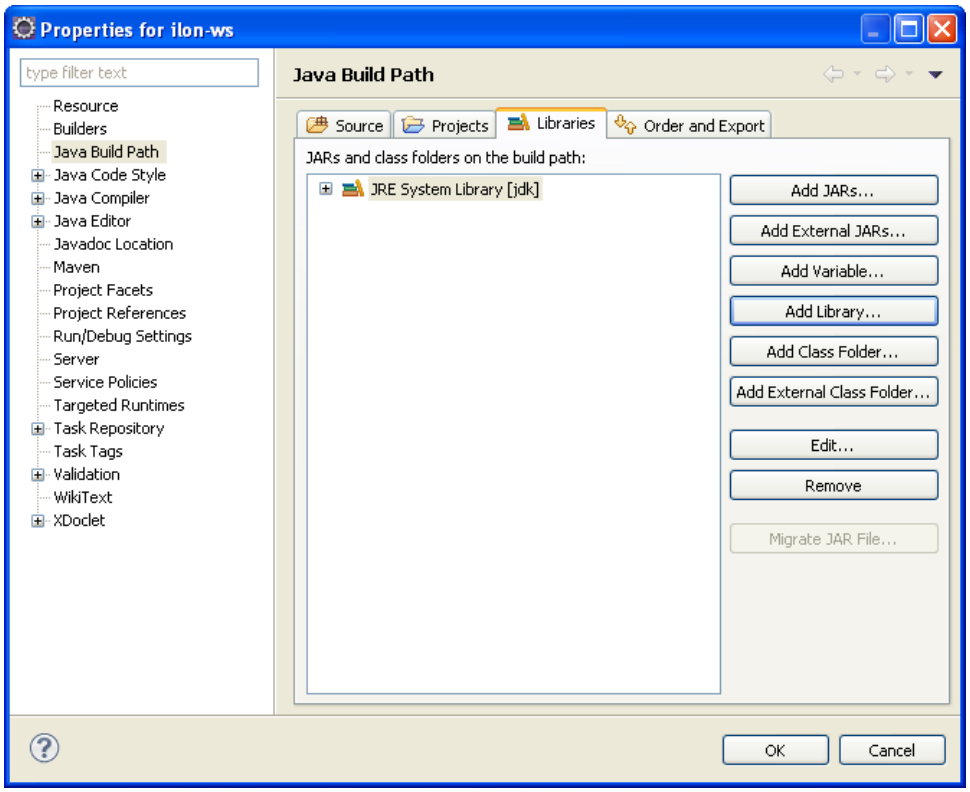
- In the **Properties** dialog, click **Java Build Path** and then click the **Libraries** tab. If there is a **JRE System Library [JavaSE-1.6]** entry listed, remove it. To remove it, click it and then click **Remove**.



- Click **Add Library**, click the **JRE System Library**, click **Next**, and then click **Finish**.



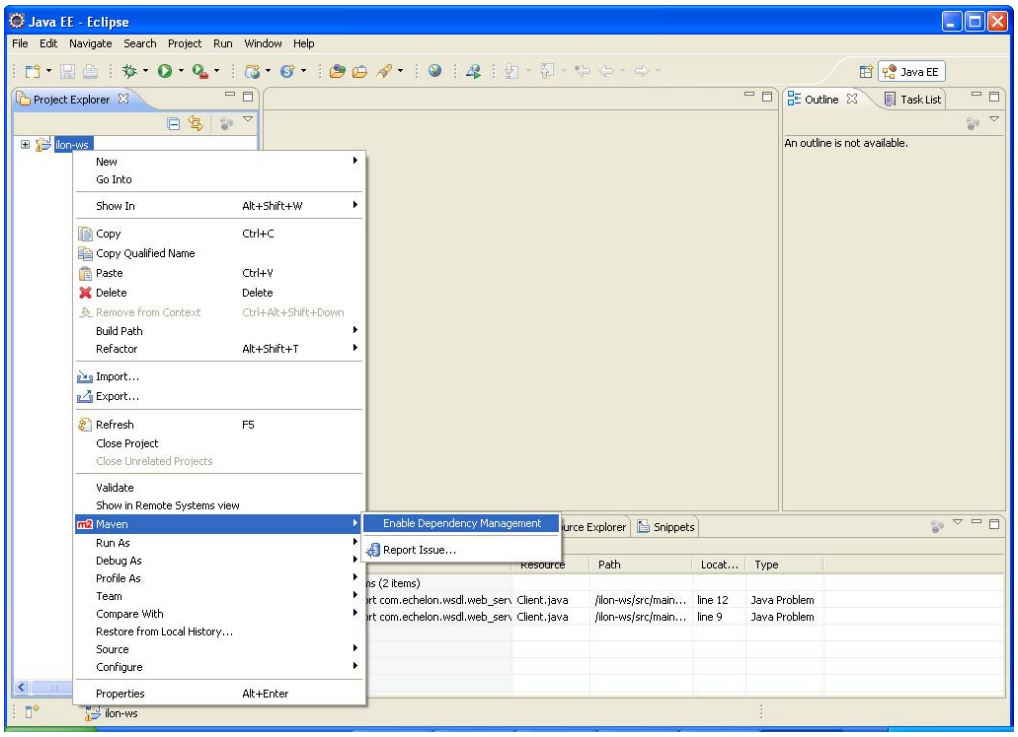
- A **JRE System Library [jdk]** entry is listed



- Click **OK**.

6. Enable Maven to manage your Java project following these steps:

a. In the **Project Explorer** view, right-click the **ilon-ws** folder, point to **Maven**, and then click **Enable Dependency Management**.



b. Browse to the **C:\eclipse\eclipse 3.5\ilon.ws.clients\jax-ws** folder, open the **.classpath** file with a text editor and change the following line:

```
<classpathentry kind="con"
path="org.eclipse.jdt.launching.JRE_CONTAINER/org.eclipse.jdt.internal.debug.u
i.launcher.StandardVMType/J2SE-1.5"/>
```

to the following:

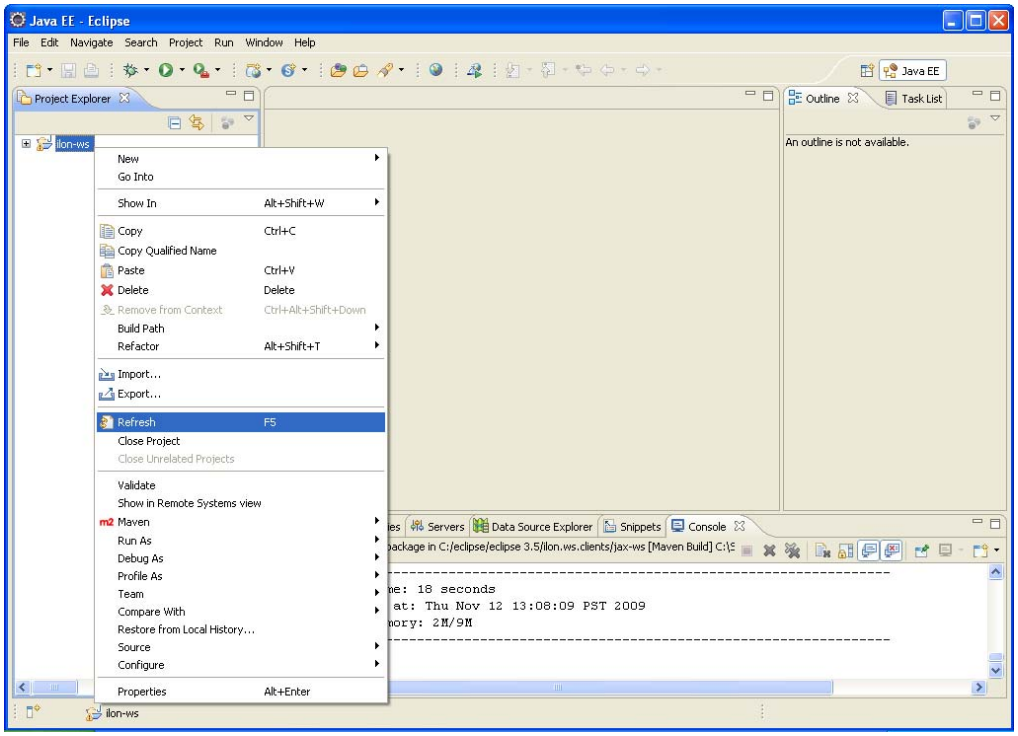
```
<classpathentry kind="con"
path="org.eclipse.jdt.launching.JRE_CONTAINER/org.eclipse.jdt.internal.debug.u
i.launcher.StandardVMType/JavaSE-1.6"/>
```

c. Restart Eclipse.

7. Generate the java proxy classes and build the jar following these steps:

a. In the **Project Explorer** view, right-click the **ilon-ws** folder, point to **Run As**, and then click **Maven Package**. This generates the java proxy classes and builds the jar.

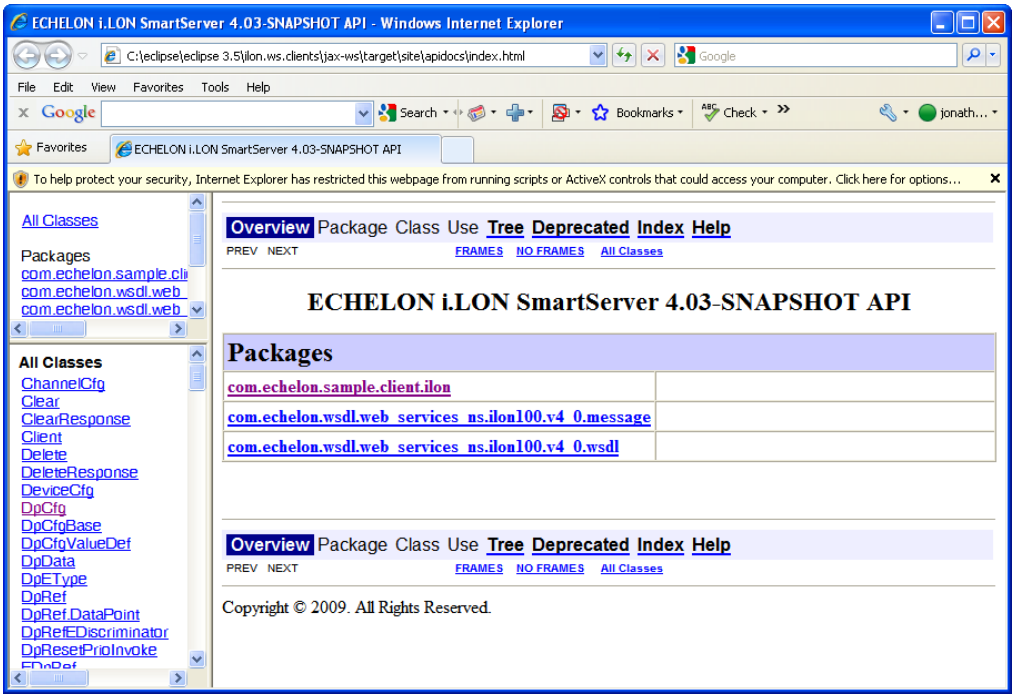
b. Refresh the **ilon-ws** folder. To do this, right-click the **ilon-ws** folder in the **Project Explorer**, and then click **Refresh**.



8. Enter the following command in the **C:\eclipse\eclipse 3.5\ilon.ws.clients** folder to generate the documentation for the example SmartServer Java project:

   mvn javadoc:javadoc

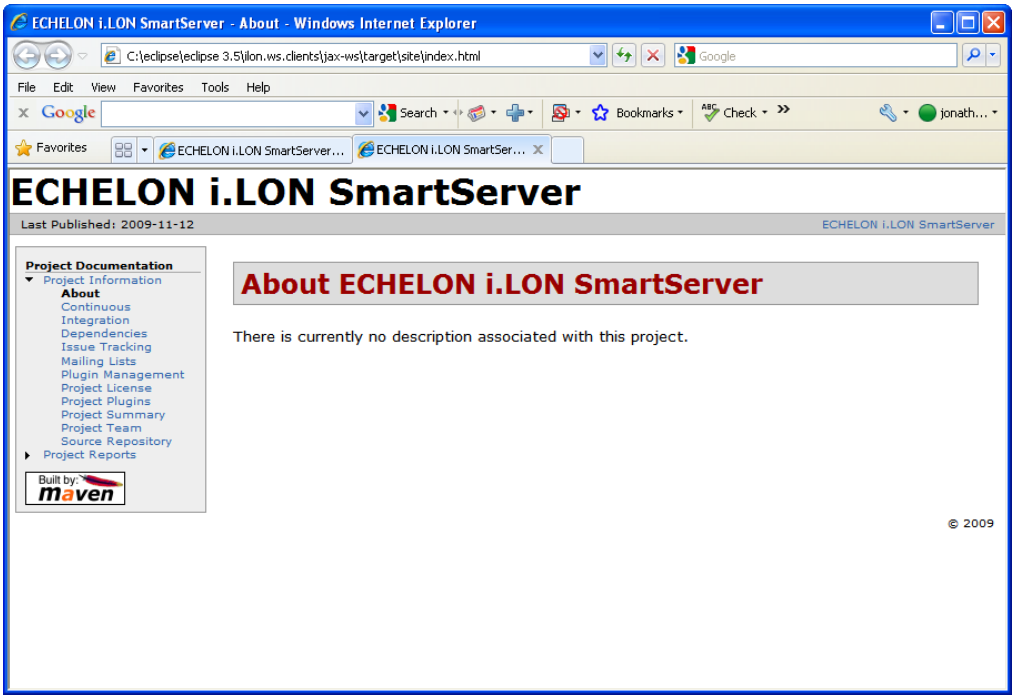
   This creates an **index.html** file in the **C:\eclipse\eclipse 3.5\ilon.ws.clients\jax-ws\target\site\apidocs** folder that you can open to view the SmartServer's API.



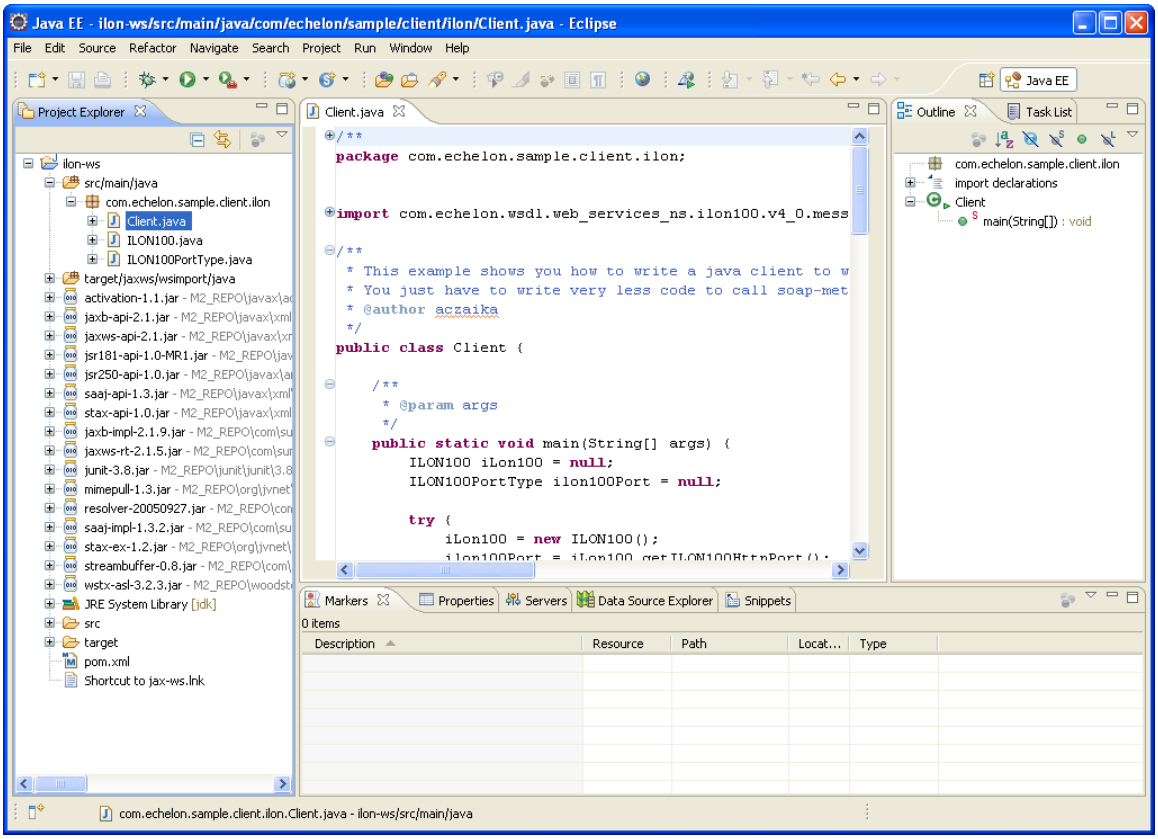
9. Enter the following command in the **C:\eclipse\eclipse 3.5\ilon.ws.clients\jax-ws** folder to generate the documentation for the project Web pages:

   mvn site

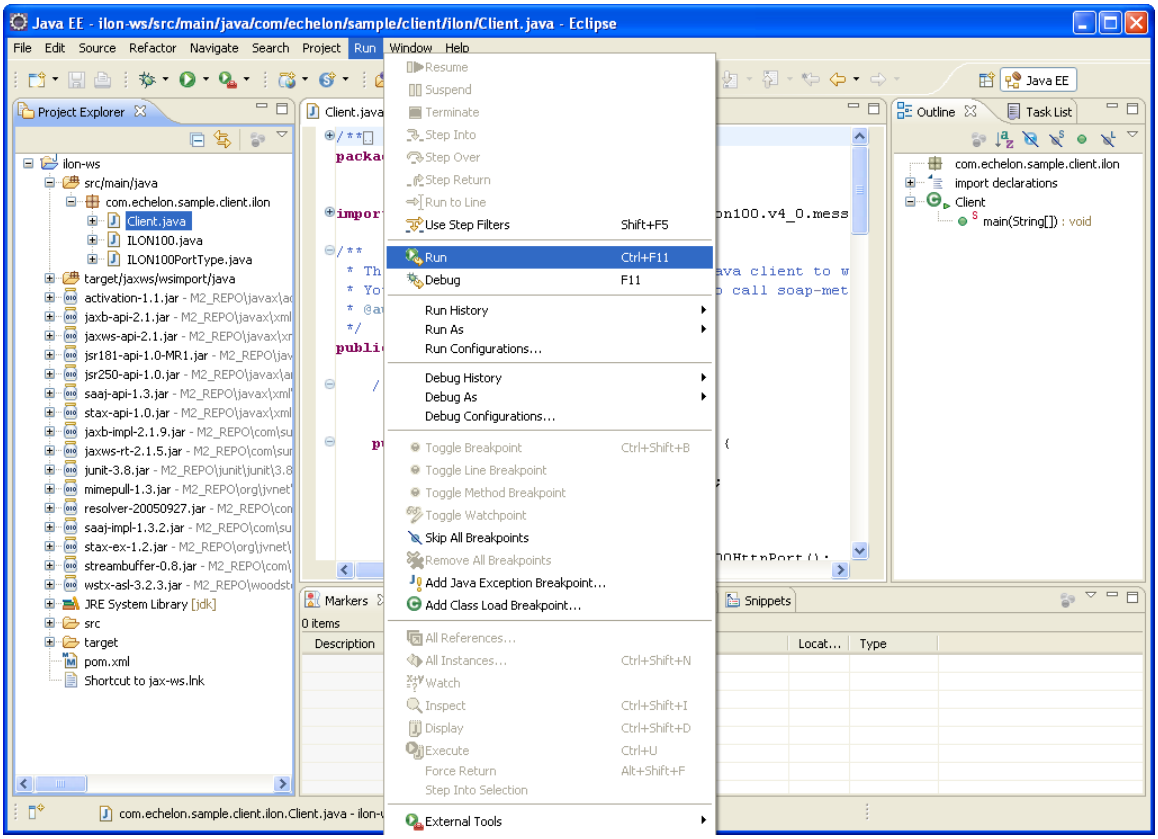
   This creates an **index.html** file in the **C:\eclipse\eclipse 3.5\ilon.ws.clients\jax-ws\target\site** folder that you can open to view the project Web pages.



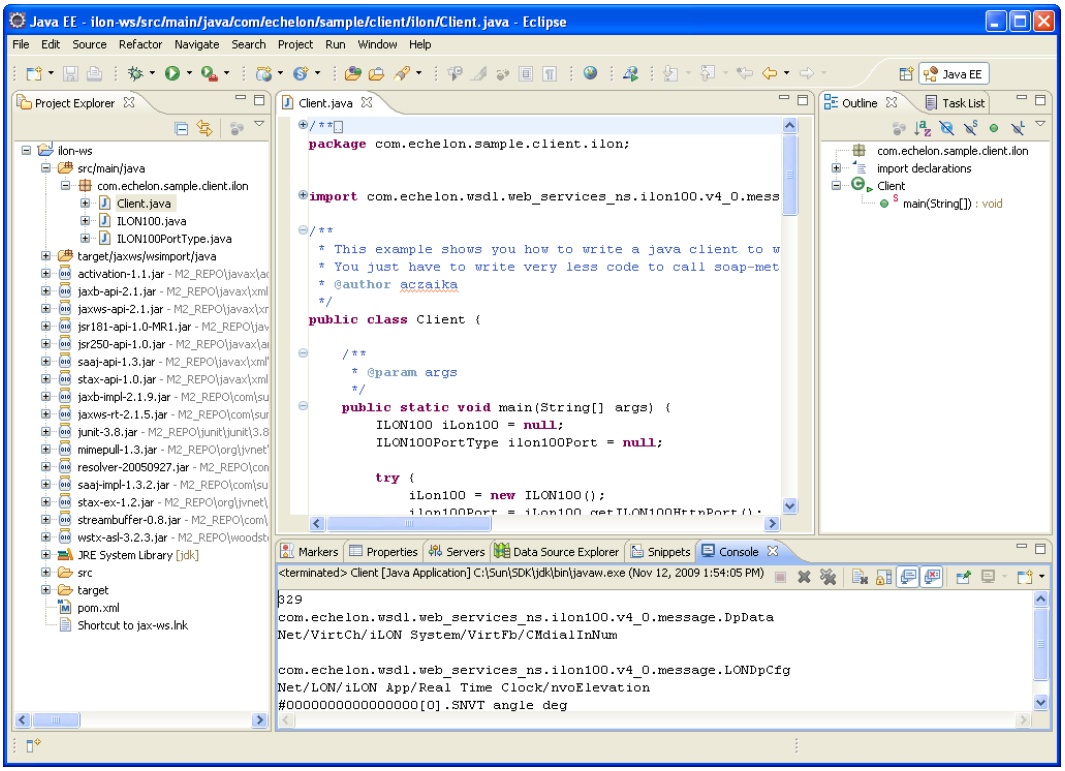
10. Open the **Client.java** class. Observe the example code in the **main()** method.



11. Run the **Client.java** class. To do this, click **Run** and then click **Run** or **Debug**.



12. Observe the output in the **Console** view at the bottom of the development environment.



22.3 Java Programming Examples

This section includes Java programming examples that demonstrate how to use the SmartServer's SOAP API to create custom applications. These programming examples create simple console applications that do the following:

- Read and write data point values.
- Create and read a data logger.
- Create and install LONWORKS devices
- Discover and install uncommissioned external device

Notes:

All examples assume that you are using a SmartServer that has been set to its factory default settings. This prevents compilation errors based on mismatching <UCPTname> properties of the objects in the LONWORKS network hierarchy (*network*/*channel*/*device*/*functional block*/*data point*).

You can download these programming examples from the *i*.LON SmartServer Community Web site at *ilonsmartserver.com*.

22.3.1 Reading and Writing Data Point Values in Java

This Java example toggles the SmartServer's digital relay outputs when run. It demonstrates how to use an xSelect statement to filter items returned by a *List*() method, and it demonstrates how to write to data points using values and presets.

You can execute this code after you have setup the Java programming environment as described in section 22.1, and created the Web service client as described in section 22.2.

For more information on the data point properties set and read in this example, see section 4.3.2, *Using the Get Function on the Data Server*, and *section 4.3.3, Using the Read Function on the Data Server*, respectively.

```
package com.echelon.sample.client.ilon;

import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.DpData;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.EXSelect;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.ItemColl;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.ItemDataColl;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.wsdl.ILON100;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.wsdl.ILON100PortType;

public class Client_DpReadWrite {

/**
* @param args
*/
public static void main(String[] args) {
ILON100 iLon100 = null;
ILON100PortType SmartServer = null;

try {
    iLon100 = new ILON100();
    SmartServer = iLon100.getILON100HttpPort();

    try {
        // ____________________________
        // Soap::List
        EXSelect xSelect = new EXSelect();
        xSelect.setXSelect("//Item[@xsi:type=\"Dp_Cfg\"]
          [contains(UCPTaliasName,\"nviClaValue\")]");
        ItemColl itemColl = SmartServer.list(xSelect);

        if(0 < itemColl.getUCPTfaultCount()) {
            System.out.printf("List-Response contains %s faults\r\n",
              itemColl.getUCPTfaultCount());
        }
        // just print the returned count of Item-s
        System.out.println("Items returned = " + itemColl.getItem().size());

        if(itemColl.getItem().size()> 0) {
            // ____________________________
            // Soap::Read

            ItemDataColl itemDataColl = SmartServer.read(itemColl);

            if(0 < itemDataColl.getUCPTfaultCount()) {
                System.out.printf("Read-Response contains %s faults\r\n",
                 itemColl.getUCPTfaultCount());
            }

            // just print some properties
            for (int i = 0; i <itemColl.getItem().size(); i++)
            {
                System.out.print(((DpData)(itemDataColl.getItem().get(i))).getUCPTname()+ " = ");
                System.out.print(((DpData)(itemDataColl.getItem().get(i))).getUCPTvalue()
                  .get(0).getValue() + "(Value Read)" + "\r\n");

                DpData dpData = (DpData) itemDataColl.getItem().get(i);

                if(dpData.getUCPTvalue().get(0).getValue().compareTo ("0.0 0")== 0)
                {
                        dpData.getUCPTvalue().get(0).setValue("100.0 1");
                        dpData.getUCPTvalue().get(1).setValue("ON");
                        itemDataColl.getItem().add(dpData);
                }

                else if(dpData.getUCPTvalue().get(0).getValue().compareTo ("100.0 1")== 0)
                {
                        dpData.getUCPTvalue().get(0).setValue("0.0 0");
                        dpData.getUCPTvalue().get(1).setValue("OFF");
                        itemDataColl.getItem().add(dpData);
                }
```

```
                }

                ItemColl writeResponse = SmartServer.write(itemDataColl);

                if(writeResponse.getUCPTfaultCount()> 0)
                {
                    // print out error and exit
                    System.out.println("An error occurred:");

                    for (int j = 0; j <itemColl.getItem().size(); j++)
                    {
                       System.out.println("Item: " + itemColl.getItem().get(j).getUCPTname() + ",
                         fault code: " + itemColl.getItem().get(j).getFault().getFaultcode() + ",
                         fault string: " + itemColl.getItem().get(j).getFault().getFaultstring());
                    }
                }

                else
                 {
                     // success
                    System.out.println("\r\n" + "Write is successful");

                     for (int j = 0; j <itemColl.getItem().size(); j++)
                     {
                           System.out.print(((DpData)(itemDataColl.getItem().get(j))).getUCPTname()+
                             " = ");

                           System.out.print(((DpData)(itemDataColl.getItem().get(j))).getUCPTvalue().
                           get(0).getValue()+ "(Value Written)" + "\r\n");
                     }
                }
              }
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
    catch (Exception e) {
        System.out.println(e.getMessage());
    }
    finally {
        iLon100 = null;
        SmartServer = null;
    }
  }
}
```

22.3.2 Creating and Reading a Data Logger in Java

The following Java example creates a data logger and then reads the data recorded by it. You can execute this code after you have setup the Java programming environment as described in section 22.1, and created the Web service client as described in section 22.2.

22.3.2.1 Creating a Data Logger

The following Java example creates a new data logger from an existing uninstantiated (hidden) data logger on the SmartServer, specifies the type, format, and size of the new data logger, and then specifies that the data logger record both of the SmartServer's digital relay outputs every minute (the Net/LON/iLON App/Digital Output 1/nviClaValue_1 and Net/LON/iLON App/Digital Output 2/nviClaValue_2 data points).

```
package com.echelon.sample.client.ilon;

import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.ELonString;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.EXSelect;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.Item;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.ItemCfgColl;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.ItemColl;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.LONFbCfg;
```

```
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.UFPTdataLoggerCfg;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.UFPTdataLoggerDpRef;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.wsdl.ILON100;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.wsdl.ILON100PortType;

public class Client_DataLoggerCreate {

/**
 * @param args
 */
  public static void main(String[] args) {
    ILON100 iLon100 = null;
    ILON100PortType SmartServer = null;

     try {
        iLon100 = new ILON100();
        SmartServer = iLon100.getILON100HttpPort();

        try {

            // ____________________________
            // Soap::List

            //Create LON Fb for the Data Logger

            EXSelect xSelect = new EXSelect();
            xSelect.setXSelect("//Item[@xsi:type=\"LON_Fb_Cfg\"]
              [starts-with(UCPTname,\"Net/LON/iLON App/Data\")][UCPThidden = \"1\"]");
            ItemColl itemColl = SmartServer.list(xSelect);


            if(0 < itemColl.getUCPTfaultCount())
            {
                System.out.println("List-Response contains " + itemColl.getUCPTfaultCount() + " faults");
            }

            itemColl.setXSelect("//Item[@xsi:type=\"LON_Fb_Cfg\"]");
            ItemCfgColl  itemCfgColl = SmartServer.get(itemColl);

            if(0 < itemCfgColl.getUCPTfaultCount())
            {
                System.out.println("List-Response contains " + itemCfgColl.getUCPTfaultCount() + " faults");
            }


             //specify Data Logger name for LonFb name
            itemCfgColl.getItem().get(0).setUCPThidden((short)(0));
            itemCfgColl.getItem().get(0).setUCPTname("Net/LON/iLON App/myDataLogger");
            ItemColl itemColl_SetReturn = SmartServer.set(itemCfgColl);

             Item myLonFb = itemColl_SetReturn.getItem().get(0);
             System.out.println("Successfully created the following LonFb = " + myLonFb.getUCPTname());

            //create new UFPTDataLogger

             UFPTdataLoggerCfg myDataLogger = new UFPTdataLoggerCfg();
             myDataLogger.setUCPTname("Net/LON/iLON App/myDataLogger");
             myDataLogger.setUCPTannotation("8000010128000000[4].UFPTdataLogger");
             myDataLogger.setUCPTlogFileName("Net/LON/iLON App/myDataLogger.csv");
             myDataLogger.setUCPTlogSize(100);
             myDataLogger.setUCPTlogLevelAlarm(50);

             //set Data Log Type
             ELonString logType_LonString = new ELonString();
             logType_LonString.setValue("LT_HISTORICAL");
             logType_LonString.setLonFormat("UCPTlogType");
             myDataLogger.setUCPTlogType(logType_LonString);

             //set Data Log Format
             ELonString logFormat_LonString = new ELonString();
             logFormat_LonString.setValue("LF_TEXT");
```

```
            logFormat_LonString.setLonFormat("UCPTlogFormat");
            myDataLogger.setUCPTlogFormat(logFormat_LonString);

            //specify two data points to be logged by new Data Logger
            UFPTdataLoggerDpRef dataPointRef1 = new UFPTdataLoggerDpRef();
            dataPointRef1.setUCPTname("Net/LON/iLON App/Digital Output 1/nviClaValue_1");
            dataPointRef1.setUCPTformatDescription ("0000000000000000[0].SNVT_switch");
            dataPointRef1.setUCPTpollRate(60);
            dataPointRef1.setDpType("Input");

            UFPTdataLoggerDpRef dataPointRef2 = new UFPTdataLoggerDpRef();
            dataPointRef2.setUCPTname("Net/LON/iLON App/Digital Output 2/nviClaValue_2");
            dataPointRef2.setUCPTformatDescription ("0000000000000000[0].SNVT_switch");
            dataPointRef2.setUCPTpollRate(60);
            dataPointRef2.setDpType("Input");

            //add the two data points to the Data Logger
            myDataLogger.getDataPoint().add(dataPointRef1);
            myDataLogger.getDataPoint().add(dataPointRef2);

            //call Set function
            ItemCfgColl itemCfgColl_DataLogger_Set = new ItemCfgColl();
            itemCfgColl_DataLogger_Set.getItem().add(myDataLogger);
            ItemColl itemColl_Set_DataLogger_Return = SmartServer.set(itemCfgColl_DataLogger_Set);

            if (itemColl_Set_DataLogger_Return.getUCPTfaultCount() > 0)
            {
               System.out.println("Set-Response contains " +
                itemColl_Set_DataLogger_Return.getUCPTfaultCount() + "faults");
            }

            else
            {
               //Check whether Data Logger was created
               Item newDataLogger = itemColl_Set_DataLogger_Return.getItem().get(0);
               System.out.println("Successfully created the following UFPT Data Logger = " +
                       newDataLogger.getUCPTname());
            }

             //*/


           }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
    catch (Exception e) {
        System.out.println(e.getMessage());
    }
    finally {
        iLon100 = null;
        SmartServer = null;
    }
  }
}
```

22.3.2.2 Reading a Data Logger

The following Java example reads and prints out the last 10 entries for one of the two data points recorded by the new data logger you created in the previous section, *Creating a Data Logger*. For more information on the data logger properties used in this example, see section 5.3.4, *Using the Read Function on a Data Logger*.

```
package com.echelon.sample.client.ilon;

import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.DpData;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.EXSelect;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.ItemColl;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.ItemDataColl;
```

```
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.wsdl.ILON100;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.wsdl.ILON100PortType;

public class Client_DpReadWrite {

/**
 * @param args
 */
  public static void main(String[] args) {
    ILON100 iLon100 = null;
    ILON100PortType SmartServer = null;

    try {
        iLon100 = new ILON100();
        SmartServer = iLon100.getILON100HttpPort();

        try {
            // _____________________________
            // Soap::List
            EXSelect xSelect = new EXSelect();
            xSelect.setXSelect("//Item[@xsi:type=\"Dp_Cfg\"]
                                        [contains(UCPTaliasName,\"nviClaValue\")]");
            ItemColl itemColl = SmartServer.list(xSelect);

            if(0 < itemColl.getUCPTfaultCount()) {
                System.out.printf("List-Response contains %s faults\r\n",
                  itemColl.getUCPTfaultCount());
            }
            // just print the returned count of Item-s
            System.out.println("Items returned = " + itemColl.getItem().size());

            if(itemColl.getItem().size()> 0) {
                // _____________________________
                // Soap::Read

                ItemDataColl itemDataColl = SmartServer.read(itemColl);

                if(0 < itemDataColl.getUCPTfaultCount()) {
                        System.out.printf("Read-Response contains %s faults\r\n",
                         itemColl.getUCPTfaultCount());
                }

                // just print some properties
                for (int i = 0; i <itemColl.getItem().size(); i++)
                {
                        System.out.print(((DpData)(itemDataColl.getItem().get(i))).getUCPTname()+ " = ");
                        System.out.print(((DpData)(itemDataColl.getItem().get(i))).
                          getUCPTvalue().get(0).getValue()+ "(Value Read)" + "\r\n");

                        DpData dpData = (DpData) itemDataColl.getItem().get(i);

                        if(dpData.getUCPTvalue().get(0).getValue().compareTo ("0.0 0")== 0)
                        {
                                dpData.getUCPTvalue().get(0).setValue("100.0 1");
                                dpData.getUCPTvalue().get(1).setValue("ON");
                                itemDataColl.getItem().add(dpData);
                        }

                        else if(dpData.getUCPTvalue().get(0).getValue().compareTo ("100.0 1")== 0)
                        {
                                dpData.getUCPTvalue().get(0).setValue("0.0 0");
                                dpData.getUCPTvalue().get(1).setValue("OFF");
                                itemDataColl.getItem().add(dpData);
                        }

                }

                ItemColl writeResponse = SmartServer.write(itemDataColl);

                if(writeResponse.getUCPTfaultCount()> 0)
                {
```

```
                        // print out error and exit
                        System.out.println("An error occurred:");

                        for (int j = 0; j <itemColl.getItem().size(); j++)
                        {
                                System.out.println("Item: " + itemColl.getItem().get(j).getUCPTname() + ",
                                 fault code: " + itemColl.getItem().get(j).getFault().getFaultcode() + ",
                                 fault string: " + itemColl.getItem().get(j).getFault().getFaultstring());
                        }
                    }

                    else
                     {
                         // success
                        System.out.println("\r\n" + "Write is successful");

                         for (int j = 0; j <itemColl.getItem().size(); j++)
                         {
                                System.out.print(((DpData)(itemDataColl.getItem().get(j))).getUCPTname()+ " = ");
                                System.out.print(((DpData)(itemDataColl.getItem().get(j))).
                                  getUCPTvalue().get(0).getValue()+ "(Value Written)" + "\r\n");
                         }
                    }
                 }
            }
            catch (Exception e) {
                System.out.println(e.getMessage());
            }
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
        finally {
            iLon100 = null;
            SmartServer = null;
        }
      }
    }
```

22.3.3 Creating and Installing a LONWORKS Device in Java

This Java example creates two LONWORKS devices, and then it commissions the devices, starts the devices' applications, and gets the devices' templates (to display the devices' functional blocks and data points in the SmartServer Web interface). The example then prints out the names and statuses of the devices that have been installed. Note that you need to replace the values of the <UCPTname>, <UCPTuniqueID>, <UCPTprogramID>, and <UCPTurlTemplate> properties provided in this example with those of the devices you are creating and installing.

You can execute this code after you have setup the Java programming environment as described in section 22.1, and created the Web service client as described in section 22.2. You must also upload the device interface (XIF) files of the devices you are creating to the root/LonWorks/import folder on the SmartServer flash disk.

For more information on the LONWORKS device properties set in this example, see section 14.3.2, *Using the Get Function on a LonWorks Device.* For more information on the network management commands issues in this example, see section 14.3.3.1, *Issuing Network Management Commands*.

```
package com.echelon.sample.client.ilon;

import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.ELonString;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.ItemCfgColl;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.ItemColl;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.LONDeviceCfg;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.LONDeviceECommand;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.wsdl.ILON100;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.wsdl.ILON100PortType;

public class Client_LonDevice {
```

```
/**
 * @param args
 */

public static byte[] hexStringToByteArray(String s) {
    int len = s.length();
    byte[] data = new byte[len / 2];
    for (int i = 0; i < len; i += 2) {
        data[i / 2] = (byte) ((Character.digit(s.charAt(i), 16) << 4)
                             + Character.digit(s.charAt(i+1), 16));
    }
    return data;
}

public static void main(String[] args) {
    ILON100 iLon100 = null;
    ILON100PortType SmartServer = null;

    try {
        iLon100 = new ILON100();
        SmartServer = iLon100.getILON100HttpPort();

        try {
            // -------------- CREATING LONWORKS DEVICES --------------

             //Create a new LON_Device_Cfg Item

             LONDeviceCfg my_LON_Device1 = new LONDeviceCfg();
             LONDeviceCfg my_LON_Device2 = new LONDeviceCfg();

             //Create an ItemCfgColl to store the LON Devices we just created

             ItemCfgColl itemCfgColl = new ItemCfgColl();
             itemCfgColl.getItem().add(0, my_LON_Device1);
             itemCfgColl.getItem().add(1, my_LON_Device2);

             //=====CREATING AND INSTALLING LON DEVICE #1==================

             // ++++++ specify properties of new LON Device #1++++++
             my_LON_Device1.setUCPTname("Net/LON/DIO-1");
             my_LON_Device1.setUCPThidden((short)0);
             my_LON_Device1.setUCPTurlTemplate ("/root/lonWorks/Import/Echelon/LonPoint/Version3/dio-10v3.XIF");

             //set Neuron ID, which is a byte[]
             my_LON_Device1.setUCPTuniqueId(hexStringToByteArray("00a145791500"));

             //set Program ID, which is a byte[]
             my_LON_Device1.setUCPTprogramId(hexStringToByteArray("80000105288a0403"));

             //set Commission status
             ELonString commissionStatus_LonString = new ELonString();
             commissionStatus_LonString.setValue("COMMISSIONED");
             my_LON_Device1.setUCPTcommissionStatus(commissionStatus_LonString);

             //set Application status
             ELonString applicationStatus_LonString = new ELonString();
             applicationStatus_LonString.setValue("APP_RUNNING");
             my_LON_Device1.setUCPTapplicationStatus(applicationStatus_LonString);

             //++++send device commands++++++

             //commission device
             LONDeviceCfg.Command commission_my_LON_Device1 = new LONDeviceCfg.Command();
             commission_my_LON_Device1.setUCPTcommand(LONDeviceECommand.CHANGE_COMMISSION_STATUS);

             ELonString my_LON_Device1_commissionStatus = new ELonString();
             my_LON_Device1_commissionStatus.setLonFormat("UCPTstatus");
             my_LON_Device1_commissionStatus.setValue("STATUS_REQUEST");
             commission_my_LON_Device1.setUCPTstatus(my_LON_Device1_commissionStatus);

             my_LON_Device1.getCommand().add(commission_my_LON_Device1);
```

```
//run device application
LONDeviceCfg.Command setOnline_my_LON_Device1 = new LONDeviceCfg.Command();
setOnline_my_LON_Device1.setUCPTcommand(LONDeviceECommand.CHANGE_APPLICATION_STATUS);

ELonString my_LON_Device1_applicationStatus = new ELonString();
my_LON_Device1_applicationStatus.setLonFormat("UCPTstatus");
my_LON_Device1_applicationStatus.setValue("STATUS_REQUEST");
setOnline_my_LON_Device1.setUCPTstatus(my_LON_Device1_applicationStatus);

my_LON_Device1.getCommand().add(setOnline_my_LON_Device1);

//get the device template to show FBs and DPs in Web UI
LONDeviceCfg.Command getTemplate_my_LON_Device1 = new LONDeviceCfg.Command();
getTemplate_my_LON_Device1.setUCPTcommand(LONDeviceECommand.GET_TEMPLATE);

ELonString my_LON_Device1_templateStatus = new ELonString();
my_LON_Device1_templateStatus.setLonFormat("UCPTstatus");
my_LON_Device1_templateStatus.setValue("STATUS_REQUEST");
getTemplate_my_LON_Device1.setUCPTstatus(my_LON_Device1_templateStatus);

my_LON_Device1.getCommand().add(getTemplate_my_LON_Device1);

//=====CREATING AND INSTALLING LON DEVICE #2==================

// ++++++ specify properties of new LON Device #2++++++
my_LON_Device2.setUCPTname("Net/LON/DIO-2");
my_LON_Device2.setUCPThidden((short)0);
my_LON_Device2.setUCPTurlTemplate ("/root/lonWorks/Import/Echelon/LonPoint/Version3/dio-10v3.XIF");

//set Neuron ID, which is a byte[]
my_LON_Device2.setUCPTuniqueId(hexStringToByteArray("00a145784600"));

//set Program ID, which is a byte[]
my_LON_Device2.setUCPTprogramId(hexStringToByteArray("80000105288a0403"));

//set Commission status
ELonString commissionStatus_LonString_device2 = new ELonString();
commissionStatus_LonString_device2.setValue("COMMISSIONED");
my_LON_Device2.setUCPTcommissionStatus(commissionStatus_LonString);

//set Application status
ELonString applicationStatus_LonString_device2 = new ELonString();
applicationStatus_LonString_device2.setValue("APP_RUNNING");
my_LON_Device2.setUCPTapplicationStatus(applicationStatus_LonString);

//++++send device commands++++++

//commission device
LONDeviceCfg.Command commission_my_LON_Device2 = new LONDeviceCfg.Command();
commission_my_LON_Device2.setUCPTcommand(LONDeviceECommand.CHANGE_COMMISSION_STATUS);

ELonString my_LON_Device2_commissionStatus = new ELonString();
my_LON_Device2_commissionStatus.setLonFormat("UCPTstatus");
my_LON_Device2_commissionStatus.setValue("STATUS_REQUEST");
commission_my_LON_Device2.setUCPTstatus(my_LON_Device2_commissionStatus);

my_LON_Device2.getCommand().add(commission_my_LON_Device2);

//run device application
LONDeviceCfg.Command setOnline_my_LON_Device2 = new LONDeviceCfg.Command();
setOnline_my_LON_Device2.setUCPTcommand(LONDeviceECommand.CHANGE_APPLICATION_STATUS);

ELonString my_LON_Device2_applicationStatus = new ELonString();
my_LON_Device2_applicationStatus.setLonFormat("UCPTstatus");
my_LON_Device2_applicationStatus.setValue("STATUS_REQUEST");
setOnline_my_LON_Device2.setUCPTstatus(my_LON_Device2_applicationStatus);

my_LON_Device2.getCommand().add(setOnline_my_LON_Device2);

//get the device template to show FBs and DPs in Web UI
```

```
                LONDeviceCfg.Command getTemplate_my_LON_Device2 = new LONDeviceCfg.Command();
                getTemplate_my_LON_Device2.setUCPTcommand(LONDeviceECommand.GET_TEMPLATE);

                ELonString my_LON_Device2_templateStatus = new ELonString();
                my_LON_Device2_templateStatus.setLonFormat("UCPTstatus");
                my_LON_Device2_templateStatus.setValue("STATUS_REQUEST");
                getTemplate_my_LON_Device2.setUCPTstatus(my_LON_Device2_templateStatus);

                my_LON_Device2.getCommand().add(getTemplate_my_LON_Device2);

                //Call the Set() function

               ItemColl Device_Return_ItemColl = SmartServer.set(itemCfgColl);

               Device_Return_ItemColl.setXSelect("//Item[@xsi:type=\"LON_Device_Cfg\"]");

                if (Device_Return_ItemColl.getUCPTfaultCount() > 0)
                {
                    // print out error and exit
                       System.out.println("An error occurred:");

                    for (int j = 0; j < Device_Return_ItemColl.getItem().size(); j++)
                    {
                        if (Device_Return_ItemColl.getItem().get(j).getFault() != null)
                        {
                              System.out.println("Item: " +
                              Device_Return_ItemColl.getItem().get(j).getUCPTname() + ", fault code: " +
                              Device_Return_ItemColl.getItem().get(j).getFault().getFaultcode() +
                              ", fault string: " +
                              Device_Return_ItemColl.getItem().get(j).getFault().getFaultstring());
                        }
                    }
                }
                else
                {
                    itemCfgColl = SmartServer.get(Device_Return_ItemColl);

                    for (int j = 0; j < itemCfgColl.getItem().size(); j++)
                    {
                      LONDeviceCfg newDevice = (LONDeviceCfg)itemCfgColl.getItem().get(j);
                       System.out.println("New Device Created = " + newDevice.getUCPTname() +
                        ".Status = " + newDevice.getUCPTcommissionStatus().getValue()+ " and " +
                         newDevice.getUCPTapplicationStatus().getValue() + ".\r");
                    }
                }
            }
            catch (Exception e) {
                System.out.println(e.getMessage());
            }
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
        finally {
            iLon100 = null;
            SmartServer = null;
        }
    }
}
```

22.3.4 Discovering and Installing External Devices in JAVA

This Java example scans a LONWORKS network for uncommissioned devices, processes the Neuron ID and program ID data of the discovered devices, and then commissions the devices, starts the devices' applications, and gets the devices' templates (to display the devices' functional blocks and data points in the SmartServer Web interface). The example then prints out the names and statuses of the devices that have been installed.

You can execute this code after you have setup the Java programming environment as described in section 22.1, and created the Web service client as described in section 22.2. You must also upload the device interface (XIF) files of the devices you are discovering and installing to the root/LonWorks/import folder on the SmartServer flash disk, or create device templates (XML files) for the devices.

For more information on discovering uncommissioned LONWORKS devices, see section 14.1.3.2, *Issuing Network Scan Commands to Discover Devices*.

```
package com.echelon.sample.client.ilon;

import javax.xml.ws.Holder;

import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.ELonString;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.EXSelect;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.InvokeCmdResponse;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.Item;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.ItemCfg;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.ItemCfgColl;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.ItemColl;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.ItemDataColl;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.LONDeviceCfg;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.LONDeviceECommand;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.LONDeviceIlonNiECommand;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.LONNetworkEScanCommand;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.LONNetworkScanCommandInvoke;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.TemplateManagerSurrogateCfg;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.message.UFPTdataLoggerData;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.wsdl.ILON100;
import com.echelon.wsdl.web_services_ns.ilon100.v4_0.wsdl.ILON100PortType;

public class Client_LonNetwork {

    /**
     * @param args
     */

    public static byte[] hexStringToByteArray(String s) {
        int len = s.length();
        byte[] data = new byte[len / 2];
        for (int i = 0; i < len; i += 2) {
            data[i / 2] = (byte) ((Character.digit(s.charAt(i), 16) << 4)
                                  + Character.digit(s.charAt(i+1), 16));
        }
        return data;
    }

    public static void main(String[] args) {
        ILON100 iLon100 = null;
        ILON100PortType SmartServer = null;

        try {
            iLon100 = new ILON100();
            SmartServer = iLon100.getILON100HttpPort();

            try {

                //create LONNetworkScanCommandInvoke item and ScanCommand attribute
                LONNetworkScanCommandInvoke networkScan = new LONNetworkScanCommandInvoke();
                networkScan.setScanCommand(LONNetworkEScanCommand.SET_SCAN);

                //***set LONNetworkScanCommandInvoke properties***

                //1. Set network UCPTname
                networkScan.setUCPTname("Net");

                //2. Set Scan Command

                //a. set scan frequency (once or continuously)
                LONNetworkScanCommandInvoke.Command scanFrequency =
```

```
  new LONNetworkScanCommandInvoke.Command();
scanFrequency.setUCPTcommand(LONDeviceIlonNiECommand.SCAN_ONCE);

//b. set scan status
ELonString scanStatus = new ELonString();
scanStatus.setLonFormat("UCPTstatus");
scanStatus.setValue("STATUS_REQUEST");
scanFrequency.setUCPTstatus(scanStatus);

//c. add scan command to LONNetworkScanCommandInvoke item
networkScan.getCommand().add(scanFrequency);

//3. Set UCPTscan
ELonString domain = new ELonString();
domain.setLonFormat("ucptScan");
domain.setValue("NST_ILON_DOMAIN");
networkScan.getUCPTscan().add(domain);

//send InvokeCmd
ItemColl itemColl = new ItemColl();
itemColl.getItem().add(networkScan);

//**note that invokeCmd requires ItemColl to be placed in Holder class**
Holder<ItemColl> holder = new Holder<ItemColl>();
holder.value = itemColl;

SmartServer.invokeCmd(holder);
System.out.println("starting scan");

//send the GetScan command to check network scan progress
LONNetworkScanCommandInvoke networkScan_Check = new LONNetworkScanCommandInvoke();
networkScan_Check.setScanCommand(LONNetworkEScanCommand.GET_SCAN);
networkScan_Check.setUCPTname("Net");

ItemColl itemColl_Check = new ItemColl();
itemColl_Check.getItem().add(networkScan_Check);

Holder<ItemColl> holder_Check = new Holder<ItemColl>();
holder_Check.value = itemColl_Check;

//Check scan status
boolean scanDone = false;
while (!scanDone)
{
        SmartServer.invokeCmd(holder_Check);

        InvokeCmdResponse scanCheck_Response = new InvokeCmdResponse();
        scanCheck_Response.setILonItem(holder_Check.value);

        LONNetworkScanCommandInvoke scanStatusCheck =
         (LONNetworkScanCommandInvoke)
         scanCheck_Response.getILonItem().getItem().get(0);

        //if the scan is done set scanDone flag to true
        if (scanStatusCheck.getCommand().get(0).getUCPTstatus().getValue().
        compareTo("STATUS_DONE")== 0)
        {
                System.out.println("Network Scan Status = " +
                 scanStatusCheck.getCommand().get(0).getUCPTstatus().getValue());
                scanDone = true;
        }

        //if the scan is not done, keep scanDone flag at false, wait 10 seconds, and check again
        else if (scanStatusCheck.getCommand().get(0).getUCPTstatus().getValue().
          compareTo("STATUS_DONE")!= 0)
        {
                System.out.println("Network Scan Status = " +
                 scanStatusCheck.getCommand().get(0).getUCPTstatus().getValue());
                Thread.sleep(10000);
        }
}
```

```
// A "<network>/#DeviceDiscovery" data logger is automatically created by the network scan
// read the Data Logger and process the data of the discovered data
UFPTdataLoggerData deviceDiscovered = new UFPTdataLoggerData();
deviceDiscovered.setUCPTname("Net/#DeviceDiscovery");
ItemColl itemColl_DataLog = new ItemColl();
itemColl_DataLog.setXSelect("//Item[@xsi:type=\"UFPTdataLogger_Data\"]");
itemColl_DataLog.getItem().add(deviceDiscovered);

ItemDataColl dataLogger = SmartServer.read(itemColl_DataLog);
System.out.println("Devices Discovered = " + (dataLogger.getItem().size()-1));
System.out.println("=====================================================");

ItemCfgColl itemCfgColl = new ItemCfgColl();

for (int i = 1; i < dataLogger.getItem().size(); i++)
 {
       UFPTdataLoggerData dataLoggerData =
        (UFPTdataLoggerData)dataLogger.getItem().get(i);

       if (dataLoggerData != null)
     {
         System.out.println("Device #" + i + ": Neuron ID and Program ID = " +
           dataLoggerData.getUCPTvalue().get(0).getValue());
     }
      // -------------- CREATING DISCOVERED LONWORKS DEVICES--------------

         //Create a new LON_Device_Cfg Item

         LONDeviceCfg my_LON_Device = new LONDeviceCfg();
         itemCfgColl.getItem().add(my_LON_Device);

         //parse Neuron ID and Program ID from Data Logger
         String NID_PID = dataLoggerData.getUCPTvalue().get(0).getValue();
         String NID = NID_PID.substring(0, 12);
         String PID = NID_PID.substring(13, 29);
         System.out.println("Neuron ID = " + NID);
         System.out.println("Program ID = " + PID);

         //set Neuron ID, which is a byte[]
         my_LON_Device.setUCPTuniqueId(hexStringToByteArray(NID));

         //set Program ID, which is a byte[]
         my_LON_Device.setUCPTprogramId(hexStringToByteArray(PID));

         //set template
          EXSelect xSelect = new EXSelect();
          xSelect.setXSelect("//Item[@xsi:type=\"TemplateManager_Cfg\"]
                              [UCPTfileType=\"TEMPLATE_OR_XIF\"]
                              [UCPTprogramId=\"" + PID + "\"]");

       itemColl = SmartServer.list(xSelect);

       TemplateManagerSurrogateCfg template =
        (TemplateManagerSurrogateCfg) itemColl.getItem().get(0);
       String templateName = template.getUCPTname();
       System.out.println("Device Template = " + templateName);

       my_LON_Device.setUCPTurlTemplate(templateName);

       //set the device name

       //1. get the name of the channel ("Net/LON")
       xSelect.setXSelect("//Item[@xsi:type=\"LON_Channel_Cfg\"]
                                 [UCPThidden=0]");
       itemColl = SmartServer.list(xSelect);

       Item channel =  itemColl.getItem().get(0);

       //2. get the name of the xif
       String[] templateName_justxif = templateName.split("/");
       int templateNameLength = templateName_justxif.length;
```

```
      String xifName = templateName_justxif[templateNameLength-1];
      System.out.println("XIF Name = " + xifName);

      //3. name device using channel name, /device [index], and xif name
      //("Net/LON/Device 1 (ai-10v3.xif)")

      String deviceName = channel.getUCPTname() + "/" + "Device " + i + "
       (" + xifName  + ")";
      System.out.println("Device Name = " + deviceName);
      System.out.println("=========================================");
      my_LON_Device.setUCPTname(deviceName);

      //set Commission status
     ELonString commissionStatus_LonString = new ELonString();
     commissionStatus_LonString.setValue("COMMISSIONED");
     my_LON_Device.setUCPTcommissionStatus(commissionStatus_LonString);

      //set Application status
    getTemplate_my_LON_Device);
}


//Call the Set() function

ItemColl Device_Return_ItemColl = SmartServer.set(itemCfgColl);

Device_Return_ItemColl.setXSelect("//Item[@xsi:type=\"LON_Device_Cfg\"]");

 if (Device_Return_ItemColl.getUCPTfaultCount() > 0)
 {
     // print out error and exit
       System.out.println("An error occurred:");

     for (int j = 0; j < Device_Return_ItemColl.getItem().size(); j++)
     {
         if (Device_Return_ItemColl.getItem().get(j).getFault() != null)
         {
              System.out.println("Item: " +
                Device_Return_ItemColl.getItem().get(j).getUCPTname() + ",
                fault code: " + Device_Return_ItemColl.getItem().get(j).
                getFault().getFaultcode() + ", fault string: " +
                Device_Return_ItemColl.getItem().get(j).getFault().
                getFaultstring());
         }
     }
 }
 else
 {
      itemCfgColl = SmartServer.get(Device_Return_ItemColl);

       if (itemCfgColl.getUCPTfaultCount() > 0)
     {
         // print out error and exit
       System.out.println("An error occurred:");
     }

     for (int j = 0; j < itemCfgColl.getItem().size(); j++)
     {
       if (itemCfgColl.getItem().get(j).getFault() != null)
         {
            System.out.println("Item: " +
              itemCfgColl.getItem().get(j).getUCPTname() + ", fault code: " +
              itemCfgColl.getItem().get(j).getFault().getFaultcode() +
              ", faultstring: " + itemCfgColl.getItem().get(j).getFault().
              getFaultstring());
         }
       else
       {
              LONDeviceCfg newDevice = (LONDeviceCfg) itemCfgColl.getItem().get(j);
               System.out.println("New Device Created = " + newDevice.getUCPTname() +
               ".  Status = " + newDevice.getUCPTcommissionStatus().getValue()+ " and " +
```

```
                                newDevice.getUCPTapplicationStatus().getValue() + ".");
                        }
                    }
                }
            }

            catch (Exception e) {
                System.out.println(e.getMessage());
            }
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
        finally {
            iLon100 = null;
            SmartServer = null;
        }
    }
}
```

Appendix A: SOAP Tester Example

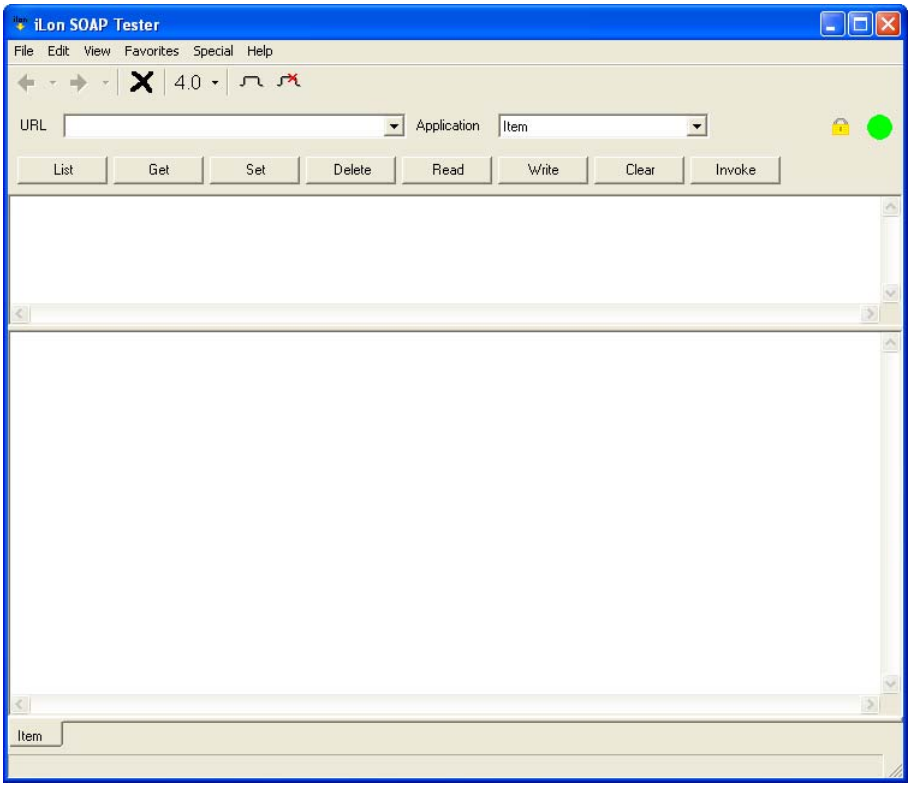
You can use the iLON SOAP Tester (version 2.0.3994) to perform functional testing of the SmartServer's pre-defined SOAP functions and your user-defined SOAP functions. The SOAP Tester is an unsupported engineering-level tool provided by Echelon. You can download the SOAP Tester from the i.LON SmartServer Community Web site at *http://ilonsmartserver.com/files*. If you have a SmartServer 2.0 (Release 4.03), the SOAP Tester is also included on the *i*.LON SmartServer 2.0 DVD in the **iLon100\iLon100\unsupported\SoapTester** folder.

This appendix provides a quick example that demonstrates how to uses the SOAP Tester to toggle the digital relay outputs on the SmartServer (the Net/LON/iLON App/Digital Output 1/nviClaValue_1 and Net/LON/iLON App/Digital Output 2/nviClaValue_2 data points). This example uses the *List*, *Get*, *Read*, and *Write* functions in the Data Server's SOAP interface.

This example assumes that you are using a SmartServer that has been set to its factory default settings. This prevents compilation errors based on mismatching <UCPTname> properties of the objects in the LONWORKS network hierarchy (*network*/*channel*/*device*/*functional block*/*data point*).

To use the SOAP Tester to toggle the digital relay outputs on the SmartServer, follow these steps:

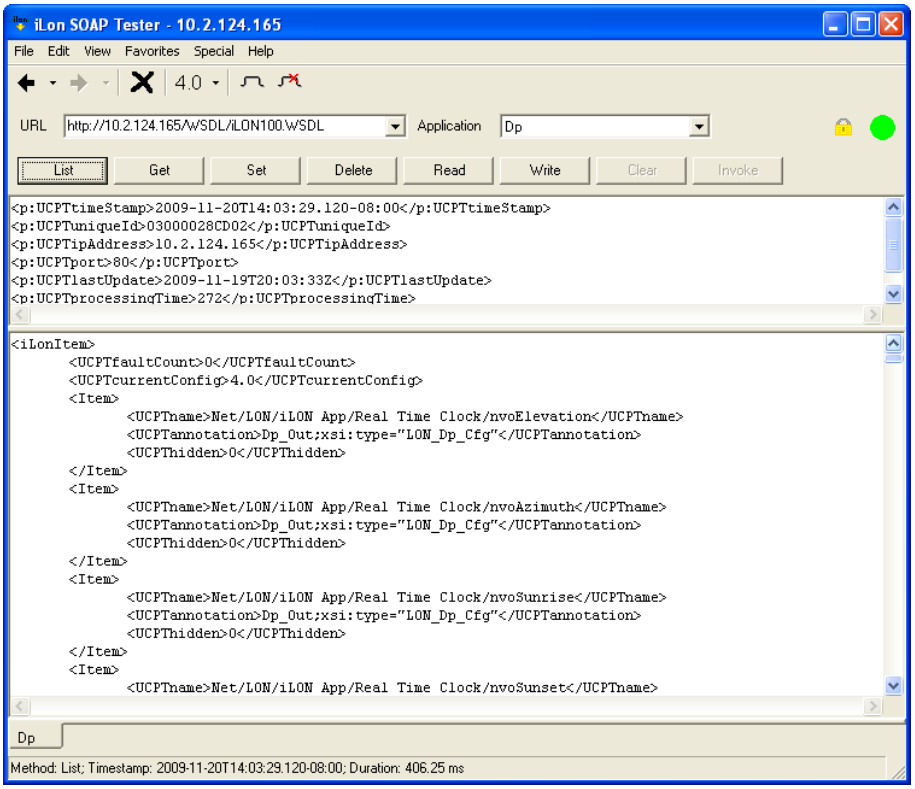
1. Start the SOAP Tester.



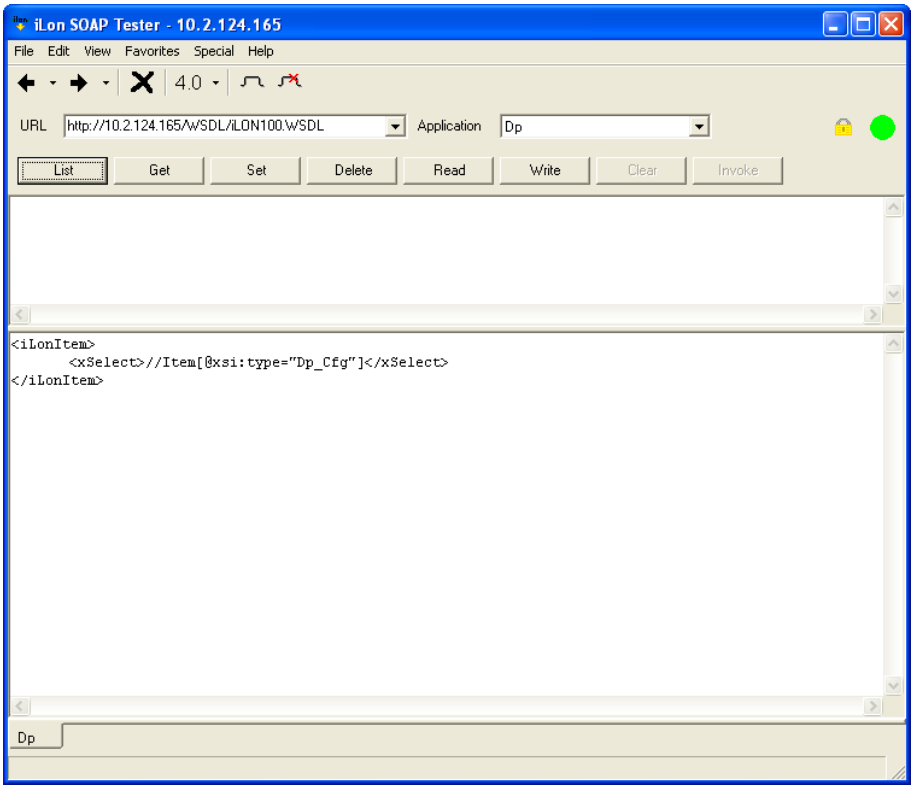
Note: By default, the SOAP Tester is set to the version **4.0** SOAP namespace used by the SmartServer. You can change the SOAP namespace to **3.0** to use the SOAP Tester with an *i*.LON *e*3 server, or you can change it to **1.1** to use the SOAP Tester with an *i*.LON *e*3 or *e*2 server.

2. In the **URL** box, enter the IP address of your SmartServer.
3. In the **Application** property, select **DP**. This sets the xsi type to **Dp_Cfg** (data point configuration). The selected xsi type will be used in the xSelect property to filter the items returned by the subsequent SOAP functions. For more information on the xsi types that you can use in xSelect statements, see section 2.5.8.1 *xsi Types*.

4. Click **List**. This calls the Data Server's *List* function and returns all the data points on the SmartServer in the SOAP response. The upper pane of the SOAP Tester displays the SOAP header, and the lower displays the SOAP body of the SOAP response.



5. Click the left arrow in the upper-left hand corner. This returns you to the *List* request used in the previous SOAP call.



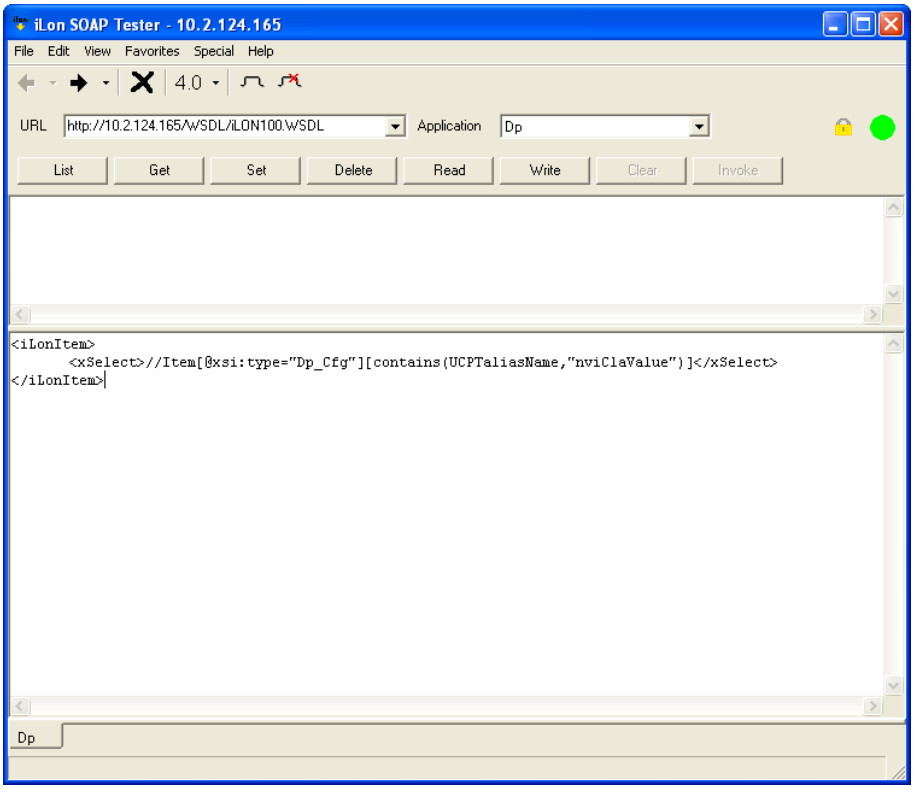
6. In this example, we want to read and write only to the SmartServer's digital relay outputs; therefore, we need to modify the *List* function so that it returns the subject data points. To do this, change the xSelect statement from:

   ```
   //Item[@xsi:type="Dp_Cfg"]
   ```

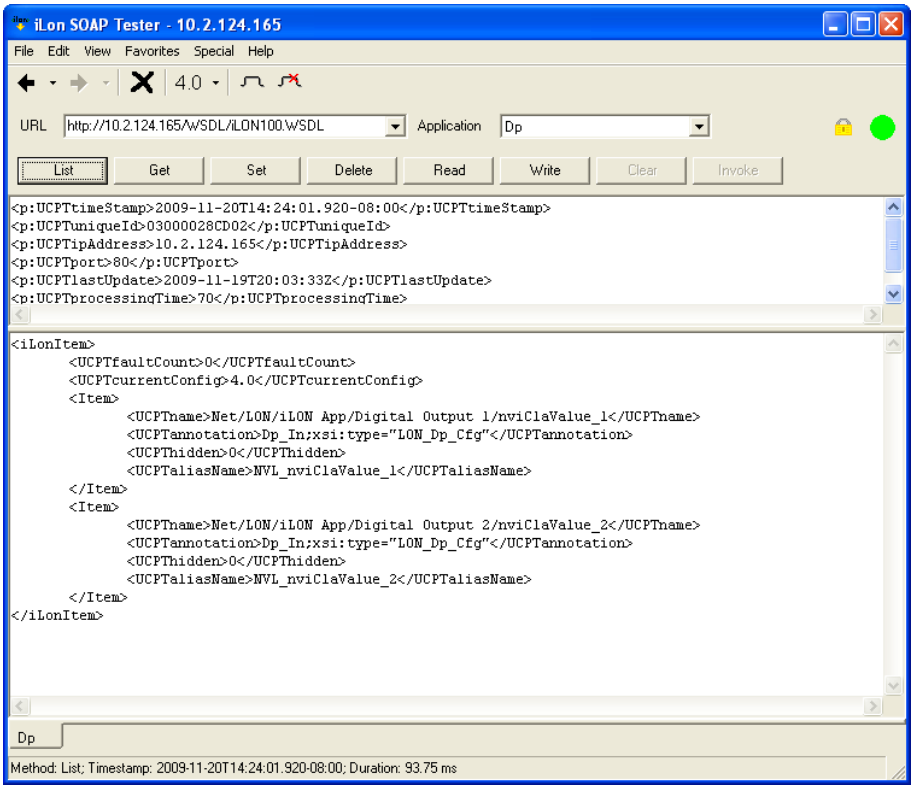
   to the following:

   ```
   //Item[@xsi:type="Dp_Cfg"][contains(UCPTaliasName,"nviClaValue")]
   ```

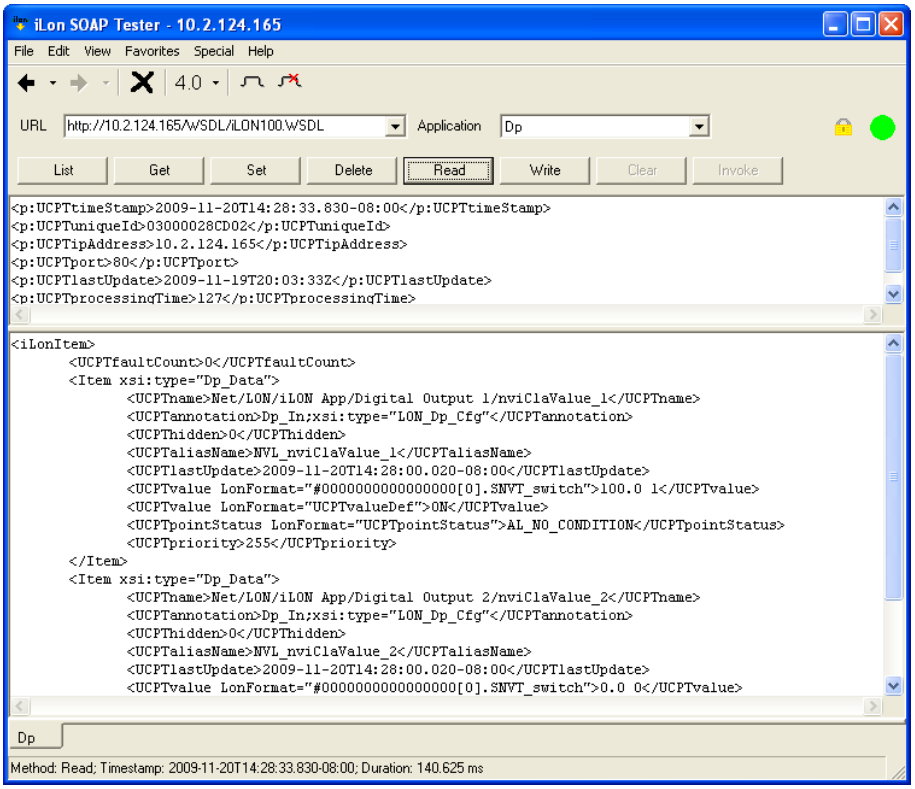
   This means that the *List* function will return only those data points that have "nviClaValue" in their alias names. The xSelect statements you will use will depend on the subject data points, as you can filter for desired items using the appropriate xsi type and predicates such as *contains* and *starts-with*.



7. Click **List**. This calls the Data Server's *List* function and returns all the data points that meet criteria specified in the xSelect statement. In this case, the Net/LON/iLON App/Digital Output 1/nviClaValue_1 and Net/LON/iLON App/Digital Output 2/nviClaValue_2 data points are returned because they are the only data points that include "nviClaValue" in their alias names (by default). For more information on using the Data Server *List* function, see section 4.3.1, *Using the List Function on the Data Server*.



8. Click **Read**. This calls the Data Server's *Read* function and returns the **Dp_Data** type for the subject data points, which includes their values, statuses, and their current priority levels. For more information on using the Data Server *Read* function, see section 4.3.4, *Using the Read Function on the Data Server*.

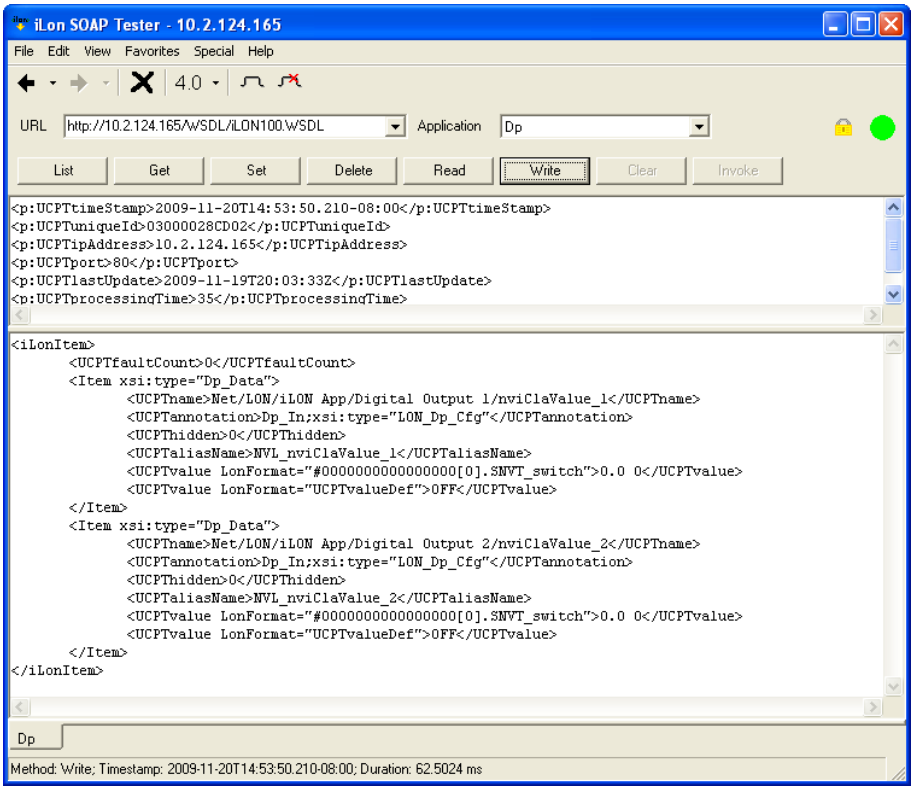


9. Toggle the values of the data points. By default, the subject data points have pre-defined ON (100.0 1) and OFF (0.0 0) presets, as specified in the **root/config/template/lonworks/Dp/#0000000000000000[0].SNVT_switch.xml** file. These presets are specified in the `<UCPTvalue LonFormat="UCPTvalueDef">` property. Because these data points have presets defined for them, you can toggle their values in the following two ways:

   - Change the preset (the `<UCPTvalue LonFormat="UCPTvalueDef">` property) to **ON** or **OFF**.
   - Delete the preset, and change the formatted value (the `<UCPTvalue LonFormat="#0000000000000000[0].SNVT_switch">` property) to **100.0 1** or **0.0 0**.

   This is because if you pass in both the <UCPTvalue> property with the formatted LonFormat attribute and a <UCPTvalue> property with the <UCPTvalueDef> LonFormat attribute in a single *Write* function, the <UCPTvalueDef> property will be used to determine the value to assign to the data point—unless it references an invalid value.

10. Click **Write**. This calls the Data Server's *Write* function and returns the updated values of the subject data points.



For more information on using the Data Server *Write* function, including how to write formatted values and presets, see section 4.3.5, *Using the Write Function on the Data Server*.

www.echelon.com