ECHELON®

LonWorks® Network XML Programmer's Guide

TURBO Edition

LNS POWERED® by ECHELON

Table of Contents

Preface

This guide describes how to create and modify a LonMaker network using the LonMaker XML Plug-in. It explains how to start the LonMaker XML Plug-in, export a LonMaker network to a LonMaker network XML file, and then import it back to create a new network or update an existing network. It provides the XML schema that defines the format of a LonMaker network XML file, and it details the actions and attributes you can specify for the elements in a LonMaker network XML file.

Purpose

This guide explains how to programmatically interface with the LonMaker XML Plug-in to update and create a LonMaker network.

Audience

This guide is intended for software developers creating applications interfacing with the LonMaker XML Plug-in. The applications may be written in any language that supports COM components or ActiveX controls, including Microsoft® Visual C# and Microsoft Visual Basic. Readers of this guide should have programming experience in such a language, and familiarity with LONWORKS technology, LNS Plug-ins, COM/ActiveX control concepts, and XML.

Related Reading

Introduction to the LONWORKS Platform—Introduces the basics of the LONWORKS platform.

LNS Plug-in Programmer's Guide—Describes how to write LNS plug-ins.

LNS Programmer's Guide—Describes the standards and development methodology for creating interoperable LNS director and plug-in applications.

LonMaker User's Guide— Describes how to use the LonMaker Integration Tool to design, commission, monitor and control, maintain, and manage a network.

Go to the LonMaker Web site at *www.echelon.com/lonmaker* for the latest versions of documentation and any available updates for your software. Go to *types.lonmark.org* for updated resource file documentation.

Content

This guide includes the following content:

- *Using the LonMaker XML Plug-in*. Explains how to write a director application that invokes the export and import functions of the LonMaker XML Plug-in. Describes the export and import automation properties you can set. Provides code samples that demonstrate how to invoke the export and import commands and how to define automation properties. Describes how to create a trace log in order to help troubleshoot your director application
- *Using the XML Schema*. Provides an overview of the XML schema that defines the structure and content of a LonMaker network XML file. Explains the elements in the header of the XML file. Describes the various object elements that may be included in the XML file and the actions that can be performed on them during import. Lists the attributes and properties you can set for each object element in a LonMaker network XML file. Provides examples of how each object element appears in a LonMaker network XML file. Describes how to define target object references. Includes a programming example that demonstrates how to export, modify, and import a LonMaker network XML file.
- *Appendix A—LonWorks Network XML Schema*. Presents the XML schema that defines the structure and content of a LonMaker network XML file.

For More Information and Technical Support

If you have technical questions that are not answered by this document, you can contact technical support. Free e-mail support is available or you can purchase phone support from Echelon or an Echelon support partner. See *www.echelon.com/support* for more information on Echelon support and training services.

You can also view free online training or enroll in training classes at Echelon or an Echelon training center to learn more about developing devices. You can find additional information about LonMaker training at *www.echelon.com/training*.

You can obtain technical support via phone, fax, or e-mail from your closest Echelon support center. The contact information is as follows (check *www.echelon.com/support* for updates to this information):

Region	**Languages Supported**	**Contact Information**
The Americas	English Japanese	Echelon Corporation Attn. Customer Support 550 Meridian Avenue San Jose, CA 95126 Phone (toll-free): 1.800-258-4LON (258-4566) Phone: +1.408-938-5200 Fax: +1.408-790-3801 *lonsupport@echelon.com*
Europe	English German French Italian	Echelon Europe Ltd. Suite 12 Building 6 Croxley Green Business Park Hatters Lane Watford Hertfordshire WD18 8YH United Kingdom Phone: +44 (0)1923 430200 Fax: +44 (0)1923 430300 *lonsupport@echelon.co.uk*
Japan	Japanese	Echelon Japan Holland Hills Mori Tower, 18F 5-11.2 Toranomon, Minato-ku Tokyo 105-0001 Japan Phone: +81.3-5733-3320 Fax: +81.3-5733-3321 *lonsupport@echelon.co.jp*
China	Chinese English	Echelon Greater China Rm. 1007-1008, IBM Tower Pacific Century Place 2A Gong Ti Bei Lu Chaoyang District Beijing 100027, China Phone: +86-10-6539-3750 Fax: +86-10-6539-3754 *lonsupport@echelon.com.cn*
Other Regions	English Japanese	Phone: +1.408-938-5200 Fax: +1.408-328-3801 *lonsupport@echelon.com*

1

Using the LonMaker XML Plug-in

This chapter explains how to write a director application that invokes the export and import functions of the LonMaker XML Plug-in. It describes the optional export and import automation properties you can set. It provides code samples that demonstrate how to invoke the export and import commands and how to define automation properties. It describes how to create a trace log in order to help troubleshoot your director application.

Introduction

You can speed up your network design and ensure your devices are configured correctly by creating a custom LonMaker user interface. Creating a custom LonMaker user interface is simple: you create a an application that reads or writes XML files and then invokes the LonMaker XML Plug-in to automatically export the LonMaker network created and import it into an LNS application. You can configure your custom LonMaker application so that it is the only user interface required to create a LonMaker network, which is ideal if the network includes only devices that are supported by your application. Alternatively, you can configure your custom application so that user interacts with it and the resulting LonMaker network (for example, the network includes devices not supported by your custom application). Writing a custom LonMaker user interface provides a simpler, quicker alternative to creating a custom LNS application. It also provides a more flexible solution than creating a Visio add-on.

To develop a custom LonMaker user interface, you need to create an application that does the following:

1. Optionally invokes the LonMaker XML Plug-in and exports all or part of a LonMaker drawing as an XML file. See *Exporting a LonMaker Network* in the following section for more information on exporting a LonMaker network.
2. Creates or modifies the XML file to create and configure the desired devices, functional blocks, network variables, and network variable/message tag connections in a LonMaker network. See *Setting Object Attributes and Properties* in Chapter 2 for more information on creating and configuring LonMaker network objects
3. Invokes the LonMaker XML Plug-in and imports the XML file created into the LonMaker tool or other LNS application. See *Importing a LonMaker Network* in the following section for more information on importing a LonMaker network.

Note: This document uses a series of C# code examples, created with Microsoft Visual Studio 2008, to demonstrate the concepts being described. You can create your custom LonMaker application using any .NET environment such as C# or Visual Basic.

Using the LonMaker XML Plug-in

You can use the LonMaker XML Plug-in to automate exporting and importing your LonMaker network to an LNS application. The LonMaker XML Plug-in provides a programmatic interface to the XML import and export features in the LonMaker tool. The LonMaker XML Plug-in is registered at the network level; therefore, it only needs to be registered once on a computer for it to be used to export or import any LonMaker network.

To use the LonMaker XML Plug-in to automate the exporting and importing of a network, you implement a director application that invokes the export or import command of the LonMaker XML Plug-in and defines the export or import automation properties. You can also define standard LNS plug-in properties in the director application.

Director applications use the LNS Plug-In API to invoke plug-in applications. The interface between director applications and plug-ins consists of standard ActiveX automation interfaces. This ActiveX-based API defines an automation object that provides a standard interface between a director application and a plug-in. Director applications can launch plug-ins and communicate with them using the methods and properties of the automation object. A set of ActiveX exceptions is defined for passing back error information from the plug-in to the director. See the *LNS Plug-In Programmer's Guide* for more information about using the LNS plug-in interface and API, and creating director applications.

Tip: You can download an assembly provided by Echelon, named **PlugInWrapper.dll**, to expedite the development of your custom LonMaker application. You can use this assembly instead of developing your own director application. To use the **PlugInWrapper.dll** file, you must have LNS Turbo Editions 3.23 (or newer) installed on your computer. The **PlugInWrapper.dll** file and the latest LNS Service Pack are available on the Echelon Web site at *www.echelon.com/downloads*.

After you download the **PlugInWrapper.dll** file, add it as a reference in your development environment. For example, if you are using Microsoft Visual Studio, click **Project**, click **Add Reference**, click **Browse**, and then select the **PlugInWrapper.dll** file. You then reference the **PlugInWrapper.dll** file in your code. You can then programmatically export and import LonMaker networks with the LonMaker XML Plug-in following the Visual C# code samples provided in the subsequent sections.

Exporting a LonMaker Network

You can use the LonMaker XML Plug-in to automate the exporting of a LonMaker network. To do this, create a director application that does the following:

1. Defines the export automation properties.
2. Invokes the export command of the LonMaker XML Plug-in.

Invoking the Export Feature

The export feature of the LonMaker XML plug-in implements the Report (23) `SendCommand` request and registers for a subsystem object, which consists of the object class (Subsystem, 5) and the subsystem name. The following Visual C# code example demonstrates how you can invoke the LonMaker XML Plug-in export command.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading;
using System.Xml;
using PlugInWrapper; //reference to Echelon PluginWrapper.dll file.
                     //You can use download and reference this instead of
                     //creating your own director application.

namespace myLmXmlNetwork
{
    class myLmXmlNetwork
    {
        static void Main(string[] args)
        {
            //Create an instance of the LonMaker plug-in using its registered
             //ActiveX name, which is "EchelonLonMakerXML.Application"
            LonMakerXmlPlugIn m_lmXml = new LonMakerXmlPlugIn();

            //Set the network name property to open the network
            m_lmXml.NetworkName = "MyXmlNetwork";

            //Invoke the send command to specify the action and target
            //object.  The send command uses the following syntax:
            //(23, 5, “network/system/subsystem[/subsystem…])”

            m_lmXml.SendCommand(23, 5, "MyXmlNetwork/MyXmlNetwork/Subsystem 1");

            //Make plug-in visible so it shows itself
            m_lmXml.Visible = true;
        }
    }
}
```

Defining Export Automation Properties

You can define standard LNS Plug-in properties and the following export automation properties in the director application. See Appendix B of the *LNS Plug-In Programmer's Guide* for more information about the standard plug-in properties you can define.

Property Name	Type	Description
XmlFileName	BSTR	Specifies the path and name of the XML file to be created. If a file with the specified name already exists, it will be overwritten. If a name is not specified, the default name will be as follows: <*LM DB Dir* >\XML\<*network name*>\Export[_*index*].XML. <*LM DB Dir*> is the LonMaker drawing and database directory. Typically, this is **C:\LM**. <*network name*> is the name of the network being exported. *index* is a decimal number appended to the file name or incremented to make the file name unique. The directory specified for the XML file must already exist.
XmlOptions	Long	Specifies export options. You can specify multiple options by ORing the following values together: • **&H01**. Suppress status dialog. The XML plug-in does not display a status dialog. The status dialog shows the progress of the export operation and any errors encountered. • **&H02**. Suppress options dialog. The XML plug-in does not display an export options dialog. If you do not set this option, you are prompted to specify the export options (including whether to cancel the export). Defaults for the options dialog are based on the parameters sent to the plug-in. • **&H10**. Export NV values. Reports the values of the NVs of all commissioned devices in the XML file. This can significantly increase the time required to export the XML file. • **&H20**. Export self-documentation. Reports self-documentation data for commissioned devices in the XML file. This can significantly increase the time required to export the XML file. • **&H40**. Export device-specific CPs. Reports the values of the device-specific CPs of commissioned devices in the XML file. This can significantly increase the time required to export the XML file.
ChannelExport	Long	Specifies which channels are reported in the exported XML file. Specify one of the following options: • **0**. Reports all channels defined in the network. This is the default. • **1**. Does not report any channels. • **2**. Reports only channels that are referenced in the

Property Name	Type	Description
		exported data.
TemplateExport	Long	Specifies which device templates are to be included in the exported XML file. Specify one of the following options: • **0**. Reports all device templates defined in the network. This is the default. • **1**. Does not report any device templates. • **2**. Reports only device templates that are referenced in the exported data.
ConnDescExport	Long	Specifies which connection description templates are to be included in the exported XML file. Specify one of the following options: • **0**. Reports all connection description templates defined in the network. This is the default. • **1**. Does not report any connection description templates. • **2**. Reports only connection description templates that are referenced in the exported data.
ExportComment	BSTR	Inserts a comment as a text string in the exported XML file.
ExportScope	Long	Specifies the subsystems to be included in the exported XML file. The application devices and routers located in the specified subsystems are included in the exported XML file. Specify one of the following options: • **0**. Reports all subsystems. Ignores the subsystem specified in the `SendCommand` request. This is the default. • **1**. Reports the subsystem specified in the `SendCommand` request and all of its nested subsystems. • **2**. Reports only the subsystem specified in the `SendCommand` request. Nested subsystems are not reported.

The following example adds code to set optional export properties.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading;
using System.Xml;
using PlugInWrapper;

namespace myLmXmlNetwork
{
    class myLmXmlNetwork
    {
        static void Main(string[] args)
        {
            LonMakerXmlPlugIn m_lmXml = new LonMakerXmlPlugIn();

            m_lmXml.NetworkName = "MyXmlNetwork";
```

```
                // *Insert export automation properties before invoking send command*

                // Option to export NV and device-specific values
                m_lmXml.XmlOptions = 0x10 | 0x40;

                //Option to export subsystem specified in SendCommand
                m_lmXml.ExportScope = 2;

                m_lmXml.SendCommand(23, 5, "MyXmlNetwork/MyXmlNetwork/Subsystem 1");
                m_lmXml.Visible = true;
            }
        }
    }
```

Importing a LonMaker Network

You can use the LonMaker XML Plug-in to automate the importing of a LonMaker network. To do this, create a director application that does the following:

1. Defines the import automation properties.
2. Invokes the import command of the LonMaker XML Plug-in.

Invoking the Import Command

The import feature of the LonMaker XML plug-in implements the user level (10000) `SendCommand` request and registers for a subsystem object, which consists of the object class (Subsystem, 5) and the subsystem name. The following Visual C# code example demonstrates how you can invoke the LonMaker XML Plug-in import command.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading;
using System.Xml;
using PlugInWrapper; //reference to Echelon PluginWrapper.dll file.
                     //You can use download and reference this instead of
                     //creating your own director application.

namespace myLmXmlNetwork
{
    class myLmXmlNetwork
    {
        static void Main(string[] args)
        {
            //Create an instance of the LonMaker plug-in using its registered
            //ActiveX name, which is "EchelonLonMakerXML.Application"
            LonMakerXmlPlugIn m_lmXml = new LonMakerXmlPlugIn();

            //Set the network name property to open the network
            m_lmXml.NetworkName = "MyXmlNetwork";

            //Invoke the send command to specify the action and target
            //object.  The send command uses the following syntax:
            //(10000, 5, “network/system/subsystem[/subsystem…])”

            m_lmXml.SendCommand(10000, 5, "MyXmlNetwork/MyXmlNetwork/Subsystem 1");

            //Make plug-in visible so it shows itself
            m_lmXml.Visible = true;
        }
    }
}
```

Defining Import Automation Properties

You can define standard LNS Plug-in properties and the following import automation properties in the director application. See Appendix B of the *LNS Plug-In Programmer's Guide* for more information about the standard plug-in properties you can define.

Property Name	Type	Description
XmlFileName	BSTR	Specifies the full path of the XML file to be imported. The file must already exist.
LogFileName	BSTR	Specifies the full path of a log file to be created during the import. The log file reports the results of the import. If a file already exists with the same name, it will be overwritten. If you do not specify a name or the field is empty, the log file is not created The directory specified for the log file must already exist.
XmlOptions	Long	Specifies export options. You can specify multiple options by ORing the following values together. • **&H01**. Suppress status dialog. The XML plug-in does not display a status dialog. The status dialog shows the progress of the import operation and any errors. • **&H02**. Suppress options dialog. The XML plug-in does not display an import options dialog. If this option is not set, you are prompted to specify the import options (including whether to cancel the import). Defaults for the options dialog are based on the parameters sent to the plug-in. • **&H04**. Import the XML data into the subsystem specified in the `SendCommand` request. If you do not set this option, the data is imported into the subsystem specified as the root subsystem in the XML file (if the root subsystem value is empty or missing, the data is imported into the first top-level subsystem in the network). The default is to import the data based on the subsystem specified in the imported XML file.

The following example adds code to set optional import properties.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading;
using System.Xml;
using PlugInWrapper;

namespace myLmXmlNetwork
{
    class myLmXmlNetwork
    {
        static void Main(string[] args)
        {
            LonMakerXmlPlugIn m_lmXml = new LonMakerXmlPlugIn();

            m_lmXml.NetworkName = "MyXmlNetwork";

            // *Insert import automation properties before invoking send command*
```

```
                // Option to suppress status and options dialog
                 m_lmXml.XmlOptions = 0x01 | 0x02;

                //Option to create and specify location of import log
                m_lmXml.LogFileName = "C:\\Lm\\XML\\XML Network\\ImportLog.XML";

                m_lmXml.SendCommand(23, 5, "MyXmlNetwork/MyXmlNetwork/Subsystem 1");
                m_lmXml.Visible = true;
            }
        }
    }
```

Troubleshooting a Director Application

You can create a trace log to verify that the LonMaker XML Plug-in is receiving the commands and options specified in your director application. This is typically only done during application development and debugging. To create a trace log for the LonMaker XML Plug-in, do the following:

1. Open the Windows registry editor. Click **Start** on the taskbar, click **Run**, type **regedit**, and then click **OK**. The **Registry Editor** dialog opens.
2. Browse to the **HKEY_LOCAL_MACHINE\SOFTWARE\LonWorks\ LonMaker for Windows** folder.
3. On the **Edit** menu, point to **New**, and then click **String Value** on the shortcut menu.
4. Type **TraceLmXml**.
5. In the right pane, right-click the **TraceLmXml** entry and then click **Modify** on the shortcut menu. The **Edit String** dialog opens.
6. In the **Value Data** box, enter **1** and then click **OK**.

The LonMaker XML Plug-in will now create a trace file in **C:\LonWorks\LonMaker\ PlugIn_LmXml.log** that reports all of the commands and options that it has received from the director application.

Creating and Updating a LonMaker Network

You can create a new network or update an existing LONWORKS network by importing a LonMaker network XML file with the LonMaker XML Plug-in. This section describes how the LonMaker XML Plug-in creates and updates networks and how it updates individual objects in the LNS network database.

How the LonMaker XML Plug-in Creates and Updates a Network

On import, the LonMaker XML Plug-in processes the data from the **RootSubsystem** specified in the XML file. If this value is not defined or it is empty, the first top-level subsystem is assumed. The plug-in then makes five passes through the XML data, performing the following actions:

1. Create, update, and delete subsystems. Deleting a subsystem also deletes all devices in that subsystem. Channels and connection descriptions are also created and updated (they are deleted on the fourth pass).
2. Create, update, and delete application devices, routers, functional blocks, and NVs/message tags. Device templates are also imported, as required.
3. Create and update NV/message tag connections.
4. Delete channels, device templates, and connection descriptions.
5. Commission application devices and routers and load the specified application image (.APB) into them.

Tip: You can import two or more individual XML files to overcome order dependencies in your XML data that are incompatible with this sequence.

How the LonMaker XML Plug-in Updates Objects

The LonMaker XML Plug-in updates objects by matching the object elements in the XML file to their corresponding objects in the LNS network database. To make a match, the plug-in first attempts to use the **NeuronID** property (devices only), provided that is has a non-zero, non-empty value; then the **Name** property; and finally the **Handle** attribute.

Once an object element in the XML file has been matched with an object in the LNS network database, the plug-in updates the properties of the object and its child objects. Only those properties that are specified in the XML file are updated (default values are not assumed). In addition, the properties of child objects are only updated if the child object itself is specified. Note that child objects are not deleted if they are not specified.

Notes:

- The plug-in will only update an object in the LNS network database if it is contained within the same scope specified in the XML file. For example, if an application device is matched using the **NeuronID** property, but the subsystem specified in the XML file for the application device does not match the one in the LNS network database, the plug-in will report an error and the application device will not be updated.
- If you specify the **CREATE** action on an object in the XML file, the plug-in will only create a new object if it cannot locate a corresponding object in the LNS network database. If the object does exist in the LNS network database, the object and its child objects are ignored.

2

Using the XML Schema

This chapter provides an overview of the XML schema that defines the structure and content of a LonMaker network XML file. It lists the actions and properties you can set for each object element in a LonMaker network XML file. It provides examples of how each object element appears in a LonMaker network XML file. It includes a programming example that demonstrates how to export, modify, and import a LonMaker network XML file.

XML Schema Overview

The LonMaker XML plug-in includes an XML schema that defines the structure and content of a LonMaker network XML file. The XML schema documents the XML file generated and read by the LonMaker XML plug-in. You can use the XML schema to validate the contents of a LonMaker network XML file to some extent. For example, the XML schema validates the length and characters for the string values of object properties; however, it does not validate the numeric and enumerated values.

The XML schema includes a header that defines the context of the XML data and object elements that correspond to objects in the LNS network database. The following sections summarize the header and object elements. All elements in the XML schema have a predefined standard or derived type to enforce LONWORKS or LNS specific constraints.

The complete XML schema is included as Appendix A. You can also view the XML schema by using an XML editor to open the **LonWorksNetwork.xsd** file in the **Echelon LonMaker XML Interface** folder. To access this folder, browse to **LonMaker\XML**in your LONWORKS directory (**C:\LonWorks** by default) or click **Start** on the taskbar, point to **Programs**, point to the **Echelon LonMaker** folder, and then click **Echelon LonMaker XML Interface**.

Header

The header of the XML schema includes the following elements listed in order of sequence:

- **Name**. Specifies the name of the LNS network database from which the XML data was exported. This field is optional upon import.
- **ReportCreated**. Specifies the data and time in which the exported XML file was created. This field is ignored upon import.
- **RootSubsystem**. Specifies the context of the subsystems, application devices, and routers in the XML file. All subsystems in the XML file must be dependents of the root subsystem. On import, if the root subsystem is empty or not specified, either the subsystem specified in the SendCommand request or the first top-level subsystem is assumed. You can use absolute or relative references to this subsystem for object elements in other subsystems such as connection targets. For absolute references, you use a leading "\" character; for relative references, you use a "**$**"character.
- **DomainID**. Specifies the domain ID of the network. If specified upon import, the domain ID of the network will be updated if it differs from the value supplied. The ID is specified as a series of 0, 2, 6, or 12 hex digits.
- **ExportScope**. Specifies the scope of the exported data.

```
<xs:element name="LonWorksNetwork">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Name" type="xs:string" minOccurs="0" />
        <xs:element name="ReportCreated" type="xs:dateTime" minOccurs="0" />
        <xs:element name="RootSubsystem" type="xs:string" minOccurs="0" />
        <xs:element name="DomainId" type="xs:string" minOccurs="0" />
        <xs:element name="ExportScope" type="enumType" minOccurs="0" />
```

Object Elements

Object elements are complex XML elements that correspond to objects in the LNS network database. Object elements include subsystems, application devices, routers, functional blocks, network variables, channels, and so on. Object elements may include a **Name** property, **Handle** attribute, **Action** attribute, and various properties. The following sections describe the **Name** property, **Handle** attribute, and **Action** attribute.

The applicable **Action** attributes and properties for each object element are detailed in the next section, *Setting Object Attributes and Properties Overview*.

```
<xs:complexType name="objectType">
   <xs:sequence />
   <xs:attribute name="Handle" type="xs:integer" use="optional" />
   <xs:attribute name="Action" type="xs:string" use="optional" />
   </xs:complexType>
```

Name

The names of an object element and its properties generally match those in the LNS network database (without the leading “Lca”). The names are different in instances where the LNS name is misleading, obsolete, or not defined.

Handle Attribute

Most object elements contain an integer **Handle** attribute that corresponds to a unique, persistent property of the object that can be used by LNS to access the object within the specified context. In many cases, the **Handle** of the object element corresponds to the LNS handle for that object; however, in some cases, the **Handle** corresponds to an index. For example, the **Handle** for functional blocks and network variables corresponds to the object’s index number within the device or functional block, respectively.

Action Attribute

Most object elements contain an enumerated, case-insensitive **Action** attribute that is applied upon import. The **Action** attribute has seven possible values:

- **CREATE**. Creates the object if it does not exist already. If it does exist, the object and all of its child objects are ignored.
- **CREATE_UNIQUE**. Forces the creation of a new object. In cases where the object name is not unique, an instance number is appended (or incremented) to make the name unique.
- **UPDATE**. Creates the object if it does not already exist; updates the object if it does exist. This is the default.
- **MODIFY**. Ignores the object if it does not already exist; updates the object if it does exist.
- **DELETE**. Deletes the object if it exists.
- **IGNORE**. Ignores the object and all of its dependents.
- **COMMISSION**. Indicates that the commissioning-related attributes of application devices or routers are to be updated. You can specify this action for subsystems, application devices, or routers. If you specify this action for any other object, the behavior is the same as if you had specified the **MODIFY** action: the object is ignored if it does not already exist or updated if it does. The commissioning action and related attributes are detailed later in this chapter.

Setting Object Attributes and Properties Overview

You can set the attributes and properties for the following object elements that may be included in a LonMaker network XML file (listed in order of level in the XML file):

- Networks
- Routers
- Subsystems
- Application devices
- Functional blocks
- Network variables
- Message tags

- Configuration properties
- Targets
- Extensions
- Channels
- Connection descriptions
- Device templates
- Component applications

For each object, individual tables that describe the applicable attributes and properties that you can set are provided. Functional block, network variable, message tag, and configuration property properties are reported within both application device and device template object elements. The properties reported within each of these object elements vary; therefore, separate property tables are provided for each object element. Properties that are reported within both object elements are described in the application device property table; the description is not repeated in the device template property table.

An example is provided after each properties table that demonstrates the structure and content of the object element.

Notes:

- Most object elements contain a **Handle** attribute that generally corresponds to the object's LNS handle; therefore, this attribute is not listed in the properties tables.
- If you use the **CREATE** action on an object element, do not specify the **Handle** attribute. This may result in unintended behavior. If you use the **CREATE_UNIQUE** action, the **Handle** attribute is ignored.
- You can use absolute or relative references for objects in one subsystem that refer to objects in another subsystem. For absolute references, you use a leading "\" character in the beginning of the subsystem path; for relative references, you use a "**$**"character. You can use forward or backward slashes as delimiters between subsystem names.
- Enumerated properties have both decimal ID and case-insensitive string values. The values appear in the following format: **<***propname* **ID= "***decimal ID***">** *string literal* **</***propname***>**. On import, the string value is checked first to determine the appropriate action. If the string value is not defined or does not match an existing value in the LNS network database, the ID is then used. If the ID is not specified, the property is not updated. The string and decimal values for enumerated properties are listed in the property description.

Networks

The network is the top level of the XML file. It includes the report options, and it contains all other object elements in the XML file.

Attributes

Networks do not have any attributes.

Properties

Property	Supported by Function?		Description
	Export	Import	
Name	Y	N	The name of the LNS network database/network.
ReportCreated	Y	N	The date and time the XML report was created. Format of date is YYYY-MM-DDThh:mm:ss
Comment	Y	N	A text string providing additional comments or description for the export.

RootSubsystem	Y	Y	The LNS Subsystem that contains all of the objects reported in the XML file. If the entire network is reported, this value is empty. On import, this value specifies the subsystem into which items are to be created; however, you can override this value.
DomainId	Y	Y	Series of 0, 2, 6, or 12 hex digits reporting the domain ID of the network.
ExportScope	Y	N	Enumerated value indicating the specified export scope. The possible values are as follows: 0 ALL 1 SUBSYSTEM_TREE 2 SUBSYSTEM 3 SELECTION
Subsystems	Y	Y	The top level subsystem containing all other subsystems, application devices, and routers. See *Subsystems* for more information.
Channels	Y	Y	See *Channels* for more information.
DeviceTemplates	Y	Y	See *Device Templates* for more information.
ConnectDescTemplates	Y	Y	See *Connection Descriptions* for more information.

Example

```
<LonWorksNetwork>
  <Name>XML Network</Name>
  <ReportCreated>2006-07-14T10:54:55</ReportCreated>
  <RootSubsystem/>
  <DomainId>31</DomainId>
  <ExportScope ID="0">ALL</ExportScope>
  <Subsystems/>
        <Subsystem/>
        <Routers/>
        <AppDevices/>
  <Channels/>
  <DeviceTemplates/>
  <ConnectDescTemplates/>
</LonWorksNetwork>
```

Subsystems

Attributes

Attribute	Applicable Values	Description/Notes
Action	1 IGNORE 2 CREATE 3 CREATE_UNIQUE 4 UPDATE 5 MODIFY 6 DELETE 7 COMMISSION	If you are creating a subsystem, you must specify the subsystem's **Name** property. If you specify the **COMMISSION** action, the subsystem properties are not updated; however, the properties of all application devices, routers, and subsystems contained within the subsystem are updated.
Ref	<*reference name*>	Allows references to the subsystem in

		NV/message tag target definitions. See *Targets* for more information on using subsystem references.

Properties

Property	Supported by Function?		Notes
	Export	**Import**	
Name	Y	Y	If you are creating a subsystem, you must specify a name in the subsystem's **Name** property that is unique within the parent subsystem.
DocumentName	Y	Y	If you do not specify this property, the subsystem will be assigned the same document name as the subsystem's parent subsystem. If the subsystem is at the root of a document, the subsystem will be assigned a document name that is the same name as the network. If the network name specified in the file matches the beginning of the specified document name, the name of the current network will be used. The maximum document name length is 80 characters.
PageName	Y	N	On import, a unique page name (within the subsystem's document) is assigned based on the subsystem name.
Location	Y	Y	The absolute path of the subsystem.
SubsystemID	Y	N	An integer value corresponding to the lower 15 bits of the LNS handle assigned to the subsystem.
Shape/StencilName	Y	Y	The name of the Visio stencil containing the master shape for the Subsystem shape. If not specified or not found, the LonBasic stencil is used on import.
Shape/MasterName	Y	Y	The name of the master shape for the Subsystem. If not specified or not found, the Subsystem shape in the LonBasic stencil is used on import.
Shape/PinX	Y	Y	The X coordinate of the Subsystem shape in the Visio page.
Shape/PinY	Y	Y	The Y coordinate of the Subsystem shape in the Visio page.
Description	Y	Y	The optional user-provided description for the subsystem.
AppDevices	Y	Y	The application devices within the subsystem. See *Application Devices* for more information.
Routers	Y	Y	The routers within the subsystem. See *Routers* for more information.
Subsystem	Y	Y	A subsystem nested within the subsystem.
Extensions	Y	Y	Extension records defined for this subsystem. See *Extension Records* for more information.

Example

```
<Subsystem Handle="6" Action="CREATE">
   <Name>Subsystem 2</Name>
   <DocumentName>XML NETWORK</DocumentName>
   <PageName>Subsystem 2</PageName>
   <Location>Subsystem 1.Subsystem 2</Location>
   <SubsystemId>6</SubsystemId>
   <Shape>
      <StencilName>LONBASIC</StencilName>
      <MasterName>Subsystem</MasterName>
      <PinX>4.890000</PinX>
      <PinY>6.300000</PinY>
   </Shape>
   <Description/>
   <Routers/>
   <AppDevices/>
   <Subsystem/>
   <Extensions>
   </Extensions>
```

Routers

Attributes

Attribute	Applicable Values	Description/Notes
Action	1 IGNORE 2 CREATE 3 CREATE_UNIQUE 4 UPDATE 5 MODIFY 6 DELETE 7 COMMISSION	If you are creating a router, you must specify the **Name** property and specify the channels attached to the router within the **NearRouterSide** and **FarRouterSide** properties. If you specify the **COMMISSION** action, the router's **Neuron ID** and **Channel** properties are updated.
Commission	0 False 1 True	You must set this property to **True** to enable commission-related operations.
State	0 DEFAULT 1 ONLINE 2 OFFLINE 3 DISABLE 4 RESTORE	You can specify the state of the router application. The default is **DEFAULT**, which indicates the settings of the LNS application are to be used.

Properties

Property	Supported by Function?		Description/Notes
	Export	Import	
Name	Y	Y	Routers are reported in the subsystem in which their shapes are defined. Routers defined in multiple subsystems are not supported. LNS requires that router names be unique within the entire network (not just within a subsystem).
Class	Y	Y	An enumerated value indicating the router type.

			The possible values are as follows: 0 ConfiguredRouter 1 LearningRouter 2 RepeaterRouter 3 BridgeRouter 4 PermanentRepeaterRouter 5 PermanentBridgeRouter If you do not specify the **Class** property when you create a router, “Configured Router” is the default router type.
Location	Y	Y	The location value reported as 12 hex digits.
LocationText	Y	Y	The location value reported as ASCII text. Non-printable ASCII characters are reported as \xHH, where HH is two hex digits. If both the **Location** and **LocationText** properties are specified, the value in the **Location** property will be used on import.
Description	Y	Y	The optional user-provided description for the router.
AuthenticationEnabled	Y	Y	An enumerated value indicating whether authentication is enabled on the device. The possible values are as follows: 0 False 1 True
InitialAuthenticationKey	Y	Y	This value is only reported on export if the value is non-empty.
Shape/StencilName	Y	Y	The name of the Visio stencil containing the master shape for the router. If you do not specify this property or it is not found on import, the LonBasic stencil is used.
Shape/MasterName	Y	Y	The name of the master shape for the router. If you do not specify this property or it is not found on import, the router shape in the LonBasic stencil is used.
Shape/PinX	Y	Y	The X coordinate of the router shape in the Visio page.
Shape/PinY	Y	Y	The Y coordinate of the router shape in the Visio page.
CommissionStatus	Y	N	An enumerated value indicating the current router configuration on export. The possible values are as follows: 0 UpdatesCurrent 1 Updates Pending 2 UpdatesFailed
AttachmentStatus	Y	N	An enumerated value indicating whether the router is communicating with the network on export. The possible values are as follows: 0 Attached 1 InImproperState 2 NotAttached

PingClass	Y	Y	An enumerated value indicating the expected movement of the router, which determines the frequency in which it is pinged. The possible values are as follows: 0 Default (default ping interval is used) 1 Mobile (1-minute ping interval is used) 2 Temporary (2-minute ping interval is used) 3 Stationary (15-minute ping interval is used) 4 Permanent (device is not pinged).
Extensions	Y	Y	Extension records defined for this router. See *Extension Records* for more information.
<RouterSide>/NeuronId	Y	Y	On export, this property is only reported if the value is non-empty. On import, the value must be unique within the network.
<RouterSide>/Channel	Y	Y	The name of the channel on which the specified side of the router is defined. To create a new router, you must specify the channel name for both sides of the router. The import operation will attempt to move a router to a new channel if either of the specified channels differ from the ones currently defined for it. The move operation will only be successfully completed if the router is unconfigured or LNS is OffNet. Each router and application device connected to a router side is moved individually. This could result in LNS errors because of missing paths for NV or MT connections.
<RouterSide>/Priority	Y	Y	On export, the current value (specified manually or selected automatically) is reported. There is no indication of how the value was selected. On import, LNS will automatically select a priority slot for the device if a value of 255 is specified.
<RouterSide>/Subnet	Y	N	The subnet element contains a **Handle** attribute which reports the subnet ID on export. The import operation does not support explicitly setting the subnet of a router.
<RouterSide>/Subnet/ Name	Y	N	The import operation does not support explicitly setting the subnet name of a router.
<RouterSide>/Subnet/ NodeId	Y	N	The import operation does not support explicitly setting the node ID of a router.

Example

```
<Routers>
   <Router Handle="2">
      <Name>RTR- 1</Name>
      <LocationText/>
```

```
        <Location>800600000000</Location>
        <Description/>
        <AuthenticationEnabled ID="0">False</AuthenticationEnabled>
        <Shape>
           <StencilName>LonPoint3</StencilName>
           <MasterName>LonPoint Router</MasterName>
           <PinX>4.510000</PinX>
           <PinY>2.620000</PinY>
        </Shape>
        <Class ID="0">ConfiguredRouter</Class>
        <PingClass ID="4">Permanent</PingClass>
        <CommissionStatus ID="1">UpdatesPending</CommissionStatus>
        <AttachmentStatus ID="0">Attached</AttachmentStatus>
        <Extensions>
        </Extensions>
        <NearRouterSide>
           <NeuronId>000000000000</NeuronId>
           <Channel>Channel 2</Channel>
           <Priority>0</Priority>
           <Subnet Handle="2">
              <Name>Subnet_1_2</Name>
              <NodeId>2</NodeId>
           </Subnet>
        </NearRouterSide>
        <FarRouterSide>
           <NeuronId>000000000000</NeuronId>
           <Channel>Channel 3</Channel>
           <Priority>0</Priority>
           <Subnet Handle="4">
              <Name>Subnet_1_3</Name>
              <NodeId>1</NodeId>
           </Subnet>
        </FarRouterSide>
     </Router>
  </Routers>
```

Application Devices

Attributes

Attribute	Applicable Values	Description/Notes
Action	1 IGNORE 2 CREATE 3 CREATE_UNIQUE 4 UPDATE 5 MODIFY 6 DELETE 7 COMMISSION	If you are creating an application device, you must specify the device's **Name**, **Device Template**, and **Channel** properties. If you specify the **COMMISSION**, the device properties are not updated, but the device's functional blocks are processed.
Ref	*<reference name>*	Allows references to the application device in NV/MT target definitions. See *Targets* for more information on using application device references.
Image	*<File Name>*	Specifies the name of the application image (.APB) to be written to the device. The file path can be absolute, or relative to an application

		image search path. If an application image is specified, the file exists, and the **Commission** attribute is set to **True**, the image is written to the device. This field is empty by default.
Commission	0 False 1 True	You must set this property to **True** to enable commission-related operations.
State	0 DEFAULT 1 ONLINE 2 OFFLINE 3 DISABLE 4 RESTORE	You can specify the state of the router application. The default is **DEFAULT**, which indicates the settings of the LNS application are to be used.
CP	0 FROM_DB 1 FROM_DEVICE 2 DEFAULTS	Specifies the handling of CP values in an application device. If you specify **DEFAULTS**, the default CP values are stored in the device template are written to the LNS network database and the application device. If you specify **FROM_DEVICE**, the current CP values stored in the application device are written to the LNS network database. If you specify **FROM_DB**, the current CP values stored in the LNS network database are written to the application device. This is the default.
DevCP	0 INCLUDE 1 EXCLUDE 2 UPLOAD 3 ONLY	Specifies the handling of device-specific CPs: If you specify **INCLUDE**, the device-specific CPs are handled using the same value specified in the **CP** attribute. If you specify **EXCLUDE**, device-specific CPs are not updated. This is the default. If you specify **UPLOAD**, device-specific CPs stored in the application device are written to the LNS network database, regardless of the value specified in the **CP** attribute If you specify **ONLY**, the device-specific CPs are handled using the same value specified in the **CP** attribute, but non–device-specific CPs are excluded.

Properties

Property	**Supported by Function?**		**Description/Notes**
	Export	**Import**	
Name	Y	Y	Application devices are reported in the subsystem in which their shapes are defined; they are not reported in the subsystems in which their functional blocks have shapes.

NeuronId	Y	Y	On export, this property is only reported if the value is non-empty. On import, the value of this property must be unique within the network.
Location	Y	Y	The location value reported as 12 hex digits.
LocationText	Y	Y	The location value reported as ASCII text. Non-printable ASCII characters are reported as **\x***HH*, where *HH* is two hex digits. If both the **Location** and **LocationText** properties are specified, the value in the **Location** property will be used on import.
Template	Y	Y	The name of the device template associated with the application device. To create a new application device on import, you must specify the device template name. If you specify a device template name that differs from the current device template, the application device will be upgraded on import and all modifications accepted (functional blocks, NVs, and connections are added or deleted, for example).
Channel	Y	Y	The name of the channel on which the application device is defined. If you specify a channel name that differs from the current channel, the import operation will attempt to move the application device to the specified channel. The move operation will only be successfully completed if the application device is unconfigured or LNS is OffNet. Application devices and routers are moved individually. This could result in LNS errors because of missing paths for NV or MT connections.
Description	Y	Y	The optional user-provided description for the application device.
SelfDocumentation	Y*	N	Self-documentation data requires communicating with the device. Self-documentation data is only reported on export if the option to report self-documentation data is set, the application device has been commissioned, and LNS is currently attached to the network.
AuthenticationEnabled	Y	Y	An enumerated value indicating whether authentication is enabled on the application device. The possible values are as follows: 0 False 1 True
InitialAuthenticationKey	Y	Y	This value is only reported on export if the value

			is non-empty.
Priority	Y	Y	On export, the current value (specified manually or selected automatically) is reported. There is no indication of how the value was selected. On import, LNS will automatically select a priority slot for the application device if you specify a value of 255.
AliasCapacity	Y	N	The number of alias table entries available on the application device.
AliasUseCount	Y	N	The number of alias table entries consumed by the application device.
Shape/StencilName	Y	Y	The name of the Visio stencil containing the master shape for the application device. If you do not specify this property or it is not found on import, the LonBasic stencil is used.
Shape/MasterName	Y	Y	The name of the master shape for the application device. If you do not specify this property or it is not found on import, the application device shape in the LonBasic stencil is used.
Shape/PinX	Y	Y	The X coordinate of the application device shape in the Visio page.
Shape/PinY	Y	Y	The Y coordinate of the application device shape in the Visio page.
CommissionStatus	Y	N	An enumerated value indicating the current device configuration on export. The possible values are as follows: 0 UpdatesCurrent 1 UpdatesPending 2 UpdatesFailed
AttachmentStatus	Y	N	An enumerated value indicating the whether the application device is communicating with the network on export. The possible values are as follows: 0 Attached 1 InImproperState 2 NotAttached
PingClass	Y	Y	An enumerated value indicating the expected movement of the application device, which determines the frequency in which it is pinged. The possible values are as follows: 0 Default (default ping interval is used) 1 Mobile (1-minute ping interval is used) 2 Temporary (2-minute ping interval is used) 3 Stationary (15-minute ping interval is used) 4 Permanent (device is not pinged).
Subnet	Y	Y	The subnet element contains a **Handle** attribute which reports the subnet ID. On import, this can be used to specify the subnet to which the device

			is to be assigned.
Subnet/Name	Y	Y	If you specify a subnet name or ID that differs from the current subnet, the import operation will attempt to move the application device to the specified subnet. If you specify both the subnet name and ID, the subnet name will first be used to look up the subnet. If the subnet is not identified, the subnet ID will be used. If you are creating an application device and specify a subnet (ID or name), the import operation will attempt to create the application device on the specified subnet.
Subnet/NodeId	Y	N	The import operation does not support explicitly setting the node ID of an application device.
Extensions	Y	Y	Extension records defined for this application device. See *Extension Records* for more information.
FunctionalBlocks	Y	Y	The functional blocks assigned to the application device that have shapes in the network are reported. The global NVs, CPs, and MTs are contained within the **FunctionalBlocks\ VrtualFunctionalBlock** property. See *Functional Blocks* for more information.

Example

```
<AppDevice Handle="1">
 <NeuronId>FEFA28FFC254</NeuronId>
 <Name>LNS Network Interface</Name>
 <LocationText/>
 <Location>000000000000</Location>
 <Template>LNS Network Interface</Template>
 <Channel>Channel 1</Channel>
 <Description/>
 <AuthenticationEnabled ID="0">False</AuthenticationEnabled>
 <Priority>0</Priority>
 <AliasCapacity>1024</AliasCapacity>
 <AliasUseCount>0</AliasUseCount>
 <Shape>
    <StencilName>LONBASIC</StencilName>
    <MasterName>Network Service Device</MasterName>
    <PinX>1.070000</PinX>
    <PinY>1.320000</PinY>
 </Shape>
 <CommissionStatus ID="1">UpdatesPending</CommissionStatus>
 <AttachmentStatus ID="0">Attached</AttachmentStatus>
 <PingClass ID="4">Permanent</PingClass>
 <Subnet Handle="1">
    <Name>Subnet_1_1</Name>
    <NodeId>127</NodeId>
 </Subnet>
```

```
  </Extensions>
  </FunctionalBlocks>
</AppDevice>
```

Functional Blocks

Attributes

Attribute	Applicable Values	Description/Notes
Action	1 IGNORE 2 CREATE 3 CREATE_UNIQUE 4 UPDATE 5 MODIFY 6 DELETE 7 COMMISSION	If you are creating a dynamic functional block, you must specify the functional block's `Name`, and **FunctionalProfile** properties. If you delete a dynamic functional block, it is removed from both the LNS network database and your network drawing on import.
Ref	*<reference name>*	Allows references to the functional block in NV/MT target definitions. See *Targets* for more information on using application device references.

Properties within Application Device Object Elements

Property	Supported by Function?		Description/Notes
	Export	Import	
Name	Y	Y	Functional blocks are reported in the subsystem in which their assigned devices are located—not in the subsystems in which their shapes are defined.
ProgrammaticName	Y	N	The original programmatic name of the functional block indicating the device object with which the functional block is associated. Virtual functional blocks do not contain this property.
FuncProfileName	Y	N	The name of the functional profile used by the functional block. Virtual functional blocks do not contain this property.
Scope	Y	N	An enumerated value indicating the scope selector of the functional block. The scope selector specifies the context in which the functional block's NV and CP values are interpreted. The possible values are as follows: -2 AutoDetermination -1 Unknown 0 Standard 1 DevClassAnyManf 2 DevClassSubClassAnyManf

			3 Manf 4 ManfDevClass 5 ManfDevClassSubClass 6 ManfDevType Virtual functional blocks do not contain this property.
IsDynamic	Y	N	An enumerated value indicating whether the functional block is static or dynamic. The possible values are as follows: 0 False (static) 1 True (dynamic)
IsVirtualFb	Y	N	An enumerated value indicating whether the functional block is a virtual functional block. 0 False (standard functional block) 1 True (virtual functional block)
Shape/StencilName	Y	Y	The name of the Visio stencil containing the master shape for the functional block. If you do not specify this property or it is not found on import, the LonBasic stencil is used.
Shape/MasterName	Y	Y	The name of the master shape for the functional block. If you do not specify this property or it is not found on import, the functional block shape in the LonBasic stencil is used.
Shape/PinX	Y	Y	The X coordinate of the functional block shape in the Visio page.
Shape/PinY	Y	Y	The Y coordinate of the functional block shape in the Visio page.
SubsystemName	Y	Y	On export, this value specifies the relative path name of the subsystem containing the functional block shape. If you are creating a functional block, you can use this property to place the functional block in a subsystem different from its associated application device. However, you cannot use this property to move an existing functional block from one subsystem to another.
Extensions	Y	Y	Extension records defined for this functional block. See *Extension Records* for more information.
NetworkVariables	Y	Y	The NVs defined on this functional block. See *Network Variables* for more information.
MTs	Y	Y	The MTs are only reported for virtual functional blocks. You can only add MTs on import to a virtual functional block.
ConfigProperties	Y	Y	The configuration properties defined on this functional block. See *Configuration Properties* for more

			information.

Example

```
<FunctionalBlock Handle="1">
   <Name>AI- 1</Name>
   <IsVirtualFb ID="0">False</IsVirtualFb>
   <ProgrammaticName>Analog Input[0]</ProgrammaticName>
   <FuncProfileName>Echelon Analog/Digital Input</FuncProfileName>
   <Scope ID="3">Manf</Scope>
   <FuncProfileProgrammaticName>UFPTAnalogDigitalInput</FuncProfileP
   rogrammaticName>
   <Shape>
     <StencilName>LonPoint3</StencilName>
     <MasterName>Analog Input</MasterName>
     <PinX>5.860000</PinX>
     <PinY>2.560000</PinY>
   </Shape>
   <IsDynamic ID="0">False</IsDynamic>
   <SubsystemName>$Subsystem 1.Subsystem</SubsystemName>
   </Extensions>
   </NetworkVariables>
   </ConfigProperties>
```

Properties within Device Template Object Elements

Property	Supported by Function?		Description/Notes
	Export	**Import**	
Name	Y	Y	
ProgrammaticName	Y	N	
FuncProfileName	Y	N	
FuncProfileProgrammatic Name	Y	N	The programmatic name given by the functional profile's LonMark definition Virtual functional blocks do not contain this property.
FuncProfileDescription	Y	N	A description of the functional profile.
PrincipalNv	Y	N	The principal NV is used to resolve the type of a CP that inherits its type. If a CP applies to a functional block and inherits its type, the CP uses the type of the principal NV.
Scope	Y	Y	
IsVirtualFb	Y	N	
ComponentApps	Y	Y	The set of LNS plug-ins registered to operate on the functional block.
NetworkVariables	Y	N	
MTs	Y	N	
ConfigProperties	Y	N	

Example

```
<FunctionalBlock Handle="2">
   <Name>Analog Input[1]</Name>
   <IsVirtualFb ID="0">False</IsVirtualFb>
   <ProgrammaticName>Analog Input[1]</ProgrammaticName>
   <FuncProfileName>Echelon Analog/Digital Input</FuncProfileName>
   <Scope ID="3">Manf</Scope>
   <FuncProfileProgrammaticName>UFPTAnalogDigitalInput</FuncProfileP
   rogrammaticName>
   <FuncProfileDescription>A generic object without feedback, used
   with any form of sensor</FuncProfileDescription>
   <PrincipalNv>AI_Analog_2</PrincipalNv>
   </NetworkVariables>
   </ConfigProperties>
```

Network Variables

Attributes

Attribute	Applicable Values	Description/Notes
Action	1 IGNORE 2 CREATE 3 CREATE_UNIQUE 4 UPDATE 5 MODIFY 6 DELETE 7 COMMISSION	You can create dynamic NVs on application devices that support them. To create a dynamic NV, you must do the following: 1. Specify the **CREATE**, **UPDATE,** or **CREATE_UNIQUE** action. 2. Set the **IsDynamic** property to **True**. 3. Specify the **Name**, **TypeSpec**, and **Direction** properties.
Ref	*<reference name>*	Allows references to the NV in target definitions. See *Targets* for more information on using references.

Properties within Application Device Object Elements

Property	Supported by Function?		Description/Notes
	Export	Import	
Name	Y	Y	The name of the NV.
Index	Y	N	The index of the NV within the application device.
LonMarkMemberIndex	Y	N	The ordinal index of the NV within its functional block.
ProgrammaticName	Y	Y	The name of this network variable as represented to the Neuron C programmer. This property is supported on import if you creating a dynamic NV.
LonMarkMemberNumber	Y	Y	A unique number within the functional block assigned by the LonMark Application Layer Interoperability Guidelines, a LonMark

			approved functional profile, or by the user (in the case of a network variable not specified in those guidelines. The LonMark Member Number can be specified when creating a dynamic NV during import; however, the value cannot be modified on an existing NV.
Direction	Y	Y	An enumerated value indicating the direction of the NV. The possible values are as follows: 0 Input 1 Output 2 Bidirectional You cannot modify this property on existing NVs.
IsTypeConfigurable	Y	N	An enumerated value indicating whether the NV type can be changed. The possible values are as follows: 0 False 1 True
TypeSpec/Scope	Y	Y	An enumerated value indicating the scope selector of the NV. The scope selector specifies the context in which the NV values are interpreted. The possible values are as follows: -2 AutoDetermination -1 Unknown 0 Standard 1 DevClassAnyManf 2 DevClassSubClassAnyManf 3 Manf 4 ManfDevClass 5 ManfDevClassSubClass 6 ManfDevType
TypeSpec/ProgramId	Y	Y	This property is only reported if the **scope ID** property has a non-zero value (the NV is non-standard).
TypeSpec/TypeName	Y	Y	You can modify the type of existing NVs by specifying a different type in the **TypeName** property, provided that the **IsTypeConfigurable** property is set to **True**. The **TypeName** must be defined in resource files for the specified scope and program ID.
Format	Y	Y	Specifies the format used by the network variable type (SI metric or US customary units, for example).
IsDynamic	Y	N	An enumerated value indicating whether the NV is static or dynamic. 0 False (static NV) 1 True (dynamic NV)

Selector	Y	N	Specifies the selector used for the NV alias.
Value	Y	N	To report NV values, you must specify the **Export NV values** option (&H10) in the **XMLOptions** export automation property (see *Invoking the Export Feature* in Chapter 1). If the NV value cannot be read from the application device, the text string “?Unknown” is reported.
HasShape	Y	Y	An enumerated value indicating whether the NV is represented by a shape on the FB shape in the network drawing. The possible values are as follows: 0 False 1 True You can preserve the NV shape on import by setting the **HasShape** property to **True**
ConnDesc	Y	N	The name of the connection description template used for connections in which the NV is the hub is reported on export. For unconnected NVs, **ConnectDescTemplate_0** is reported.
ConfigProperties	Y	Y	The NV type CPs of the NV are reported. Global CPs are reported in the **FunctionalBlocks** property. See *Configuration Properties* for more information.
Targets	Y	Y	The target NVs connected to the NV are reported. See *Targets* for more information.

Example

```
<NetworkVariable Handle="0">
   <Name>Analog</Name>
   <Index>6</Index>
   <LonMarkMemberIndex>0</LonMarkMemberIndex>
   <ProgrammaticName>AI_Analog_1</ProgrammaticName>
   <LonMarkMemberNumber>1</LonMarkMemberNumber>
   <Direction ID="1">Output</Direction>
   <IsTypeConfigurable ID="1">True</IsTypeConfigurable>
   <TypeSpec>
         <Scope ID="0">Standard</Scope>
         <TypeName>SNVT_temp_f</TypeName>
   </TypeSpec>
   <Format>SNVT_temp_f#US</Format>
   <IsDynamic ID="0">False</IsDynamic>
   <Selector>213</Selector>
   <HasShape ID="1">True</HasShape>
   <ConnDesc>SCD_Unicast</ConnDesc>
   </ConfigProperties>
   </Targets>
```

Properties within Device Template Object Elements

Property	Supported by Function?		Description/Notes
	Export	Import	
Name	Y	Y	
Index	Y	N	
LonMarkMemberIndex	Y	N	
ProgrammaticName	Y	N	
LonMarkMemberNumber	Y	N	
DirectionID	Y	N	
IsTypeConfigurable	Y	N	
TypeSpec/Scope	Y	N	
TypeSpec/ProgramId	Y	N	
TypeSpec/TypeName	Y	N	
ConnDesc	Y	N	
ConfigProperties	Y	N	There are no modifiable **ConfigProperties** properties in a device template; therefore these entries are ignored during import.

Example

```
<NetworkVariable Handle="5">
   <Name>Node_SetEELock</Name>
   <Index>5</Index>
   <LonMarkMemberIndex>-1</LonMarkMemberIndex>
   <ProgrammaticName>Node_SetEELock</ProgrammaticName>
   <LonMarkMemberNumber>-1</LonMarkMemberNumber>
   <Direction ID="0">Input</Direction>
   <IsTypeConfigurable ID="0">False</IsTypeConfigurable>
   <TypeSpec>
         <ScopeID="0">Standard</Scope>
         <TypeName>SNVT_lev_disc</TypeName>
   </TypeSpec>
<ConnDesc>ConnectDescTemplate_0</ConnDesc>
</NetworkVariable>
```

Message Tags

Attributes

Attribute	Applicable Values	Description/Notes
Action	1 IGNORE 4 UPDATE 5 MODIFY 7 COMMISSION	You cannot create or delete dynamic message tags.
Ref	*<reference name>*	Allows references to the message tag in target

			definitions. See *Targets* for more information.

Properties within Application Device Object Elements

Property	**Supported by Function?**		**Description/Notes**
	Export	**Import**	
Name	Y	Y	The name of the message tag.
Direction	Y	Y	An enumerated value indicating the direction of the message tag. The possible values are as follows: 0 Input 1 Output 2 Bidirectional You cannot modify this property on existing message tags.
AddressTableIndex			The index of the address table entry on the application devices assigned to this message tag (typically the same as the message tag index). For **msg_in**, this is reported as 65534
IsDynamic	Y	N	An enumerated value indicating whether the message tag is static or dynamic. Although, you cannot create dynamic message tags, they are reported on export and you can connect them. The possible values are as follows: 0 False (static message tag) 1 True (dynamic message tag)
HasShape	Y	Y	An enumerated value indicating whether the message tag is represented by a shape on the Virtual FB shape in the network drawing. The possible values are as follows: 0 False 1 True
ConnDesc	Y	N	The name of the connection description template used for connections in which the message tag is the hub is reported on export. For unconnected message tags, **ConnectDescTemplate_0** is reported.
Targets	Y	Y	The target MTs connected to the message tag are reported. See *Targets* for more information.

Example

```
<MessageTag Handle="0">
   <Name>msg_in</Name>
   <Direction ID="0">Input</Direction>
   <AddressTableIndex>65534</AddressTableIndex>
   <ConnDesc>ConnectDescTemplate_0</ConnDesc>
   <HasShape ID="1">True</HasShape>
   <IsDynamic ID="0">False</IsDynamic>
</MessageTag>
```

Properties within Device Template Object Elements

Property	Supported by Function?		Description/Notes
	Export	Import	
Name	Y	Y	
Direction	Y	Y	
AddressTableIndex	Y	N	
ConnDesc	Y	N	

Example

```
<MessageTag Handle="0">
   <Name>msg_in</Name>
   <Direction ID="0">Input</Direction>
   <AddressTableIndex>65534</AddressTableIndex>
   <ConnDesc>ConnectDescTemplate_0</ConnDesc>
</MessageTag>
```

Configuration Properties

Attributes

Attribute	Applicable Values	Description/Notes
Action	1 IGNORE 4 UPDATE 5 MODIFY 7 COMMISSION	You cannot create or delete CPs separately.

Properties within Application Device Object Elements

Property	Supported by Function?		Description/Notes
	Export	Import	
Name	Y	N	The name of the CP.
DeviceSpecificAttribute	Y	N	An enumerated value indicating whether the CP is a device-specific CP. The possible values are as follows: 0 False (non-device-specific CP) 1 True (device-specific CP)
ConstantAttribute	Y	N	An enumerated value indicating whether the CP is read-only. The possible values are as follows: 0 False (writable) 1 True (read-only)

Dimension	Y	N	The number of elements in the CP are reported. A value of 1 indicates a single element. A value greater than 1 indicates multiple elements (an array for example).
FlagsByte	Y	N	An enumerated value indicating the type of flag set on the CP. The possible values are as follows: 1 Disabled (the CP can only be changed when the application device is disabled). 2 Offline (the CP can only be changed when the application device is offline). 4 Constant (the CP is read-only). 8 Reset (the application device is reset when the CP value is changed). 16 ManufactureOnly (the CP can only be changed when the application device is manufactured). 32 DeviceSpecific (the value of the CP can be modified independent of the LNS network database.) An instance of this property is reported for each flag set on the CP; therefore, multiple instances of this property may be reported for a given CP. If no flags are set on the CP, a value of "None" is reported.
Value Index	Y	Y	To report the values of device-specific CPs, you must specify the **Export Device Specific CPs** option (&H40) in the **XMLOptions** export automation property (see *Invoking the Export Feature* in Chapter 1). For CP array values, each element of the array is specified as a separate **Value** property, with an **Index** attribute to indicate the array index. On import, the **Value** property is ignored if you set the **ConstantAttribute** in the device template object element to **True**.

Example

```
<ConfigProperty Handle="536870913">
   <Name>SCPTmaxRcvT</Name>
   <DeviceSpecificAttribute ID="0">False</DeviceSpecificAttribute>
   <ConstantAttribute ID="0">False</ConstantAttribute>
   <Dimension>1</Dimension>
   <FlagsByte ID="1">Disabled</FlagsByte>
   <Value Index="0">00:00:20.000</Value>
</ConfigProperty>
```

Properties within Device Template Object Elements

Property	Supported by Function?		Description/Notes
	Export	Import	
Name	Y	N	
DeviceSpecificAttribute	Y	Y	
ConstantAttribute	Y	Y	If you set the **ConstantAttribute** to **True** in the **DeviceTemplate** object element, the **Value** properties in the **ApplicationDevice** object element are ignored on import.
Dimension	Y	N	
FlagsByte	Y	N	
Value	Y	N	The value reported for CPs is the default value for the CP type—not the default value for the CP within the device template.

Example

```
<ConfigProperty Handle="268435461">
   <Name>SCPTlocation</Name>
   <DeviceSpecificAttribute ID="0">False</DeviceSpecificAttribute>
   <ConstantAttribute ID="0">False</ConstantAttribute>
   <Dimension>1</Dimension>
   <FlagsByte ID="2">Offline</FlagsByte>
   <Value Index="0"/>
</ConfigProperty>
```

Targets

Attributes

Attribute	Applicable Values	Description/Notes
Action	2 CREATE 6 DELETE	No error is reported if you specify **CREATE** but the target NV/MT is already defined, or if you specify **DELETE** and the target NV/MT does not exist.

Properties

Property	Supported by Function?		Description/Notes
	Export	Import	
ReferenceName	N	Y	References to previously defined subsystems, application devices, functional blocks, or NV/MTs are reported. See *Using References* in this section for more information. This property may be combined with properties listed below to define a complete object name,

			or override components of the referenced object.
SubsystemName	Y	Y	On export, the subsystem name is reported relative to the root subsystem of the export. See *Using References* in this section for more details. On import, the name can be specified as either an absolute path or a relative path.
DeviceName	Y	Y	Specifies the name of the application device of the target NV/message tag.
FunctionalBlockName	Y	Y	Specifies the name of the functional block of the target NV/message tag.
TargetName	Y	Y	The name of the target NV/message tag that is connected to the current hub NV/message tag or is to be added and connected to the current hub NV/message tag on import.

Example

```
<Target>
   <SubsystemName>$Subsystem1</SubsystemName>
   <DeviceName>AI-1</DeviceName>
   <FunctionalBlockName>AFB-1</FunctionalBlockName>
   <TargetName>A1</TargetName>
</Target>
```

Using References

When defining connection targets, you must specify all the properties of the target NV/message tag, which consists of the **SubsystemName**, **DeviceName**, **FunctionalBlockName**, and **TargetName** properties. If one or more of these properties is unknown, you can define object references in the target specification.

To define an object reference, you specify the **Ref** attribute with a value that can be used in connection definitions to refer back to that object. For example, to create a reference for a subsystem object, use the following syntax:

<Subsystem Action="**CREATE_UNIQUE**" Ref="**TargetSub**">.

When this XML statement is executed, the subsystem object and a reference to it named “**TargetSub**” are created. If you define a connection target, you could now specify a reference to an application device within that subsystem using the following syntax:

```
<Target>
   <ReferenceName>TargetSub</ReferenceName>
   <DeviceName>AO- 1</DeviceName>
   <FunctionalBlockName>AFB- 1</LonMarkObjectName>
   <TargetName>A1</TargetName>
</Target>
```

When using a reference name, you can override any of the **Target** properties by explicitly specifying those properties in the target specification.

Updating Connection Descriptions

You can request that the connection description used for all connections involving a specific NV/message tag hub be recomputed on import. To do this, append **RecomputeCd="True"** (case-insensitive) to the opening <Targets> tag so that the **Targets** element is as follows:

```
<Targets RecomputeCd="True">
   <Target>
```

```
         <ReferenceName>TargetSub</ReferenceName>
         <DeviceName>AO- 1</DeviceName>
         <FunctionalBlockName>AFB- 1</LonMarkObjectName>
         <TargetName>A1</TargetName>
   </Target>
</Targets>
```

The NV/message tag connection will use the existing connection description if the recompute request fails for any reason.

Extensions

Attributes

Attribute	Applicable Values	Description/Notes
Action	1 IGNORE 2 CREATE 3 CREATE_UNIQUE 4 UPDATE 5 MODIFY 6 DELETE 7 COMMISSION	To create an extension record, you must specify the **Name** property.

Properties

Property	Supported by Function?		Description/Notes
	Export	Import	
Name	Y	Y	LonMaker-defined extension records (entries in which the name begins with "Echelon_LMW") are not reported on export or imported.
CopyWithParent	Y	Y	An enumerated value indicating whether the extension record is copied with its parent record. The possible values are as follows: 0 False (not copied) 1 True (copied)
Value	Y	Y	One **Value** element is defined for each of the value properties of the Extension object. This property contains an **Index** attribute that identifies the specific property and has a value from 1 to 3. On export, only non-empty value properties are reported.
Value/Type	Y	Y	An enumerated value indicating the type of the variant data in the extension record. The possible values are as follows: 0 EMPTY N/A 1 NULL N/A 2 I2 Signed decimal 3 I4 Signed decimal

			4 R4 Floating point
			5 R8 Floating point
			6 CY Signed decimal: Unsigned decimal (high order 32 bits: low order 32 bits)
			7 DATE Floating point
			8 BSTR Character string
			9 DISPATCH Hex
			10 ERROR Signed decimal
			11 BOOL Signed decimal
			13 UNKNOWN Hex
			16 I1 Signed decimal
			17 UI1 Unsigned decimal
			18 UI2 Unsigned decimal
			19 UI4 Unsigned decimal
			22 INT Signed decimal
			23 UINT Unsigned decimal
			27 SAFEARRAY Hex digit array
			8194 ARRAY_I2 Hex digit array
			8195 ARRAY_I4 Hex digit array
			8209 ARRAY_UI1 Hex digit array
			8210 ARRAY_UI2 Hex digit array
			8211 ARRAY_UI4 Hex digit array
			8208 ARRAY_I1 Hex digit array
Value/Data	Y	Y	If the **Data** property has a non-array value, the value is an ASCII representation of the data based on the type. If the **Data** property is an array value, the data is reported as a string of hex digits (two digits per byte of data).
Value/ NumberDimensions	Y	Y	Applies to array values. Specifies the number of dimensions in the Variant type data. LNS only supports arrays with one dimension. On import, 1 is assumed if this property is not specified.
Value/ ElementSize	Y	Y	Applies to array values. Specifies the number of bytes in each element of the array.
Value/ Dimension	Y	Y	Applies to array values. A separate element is reported for each array dimension (LNS only supports arrays with one dimension; therefore, 1 is always reported). This property contains an **Index** attribute that specifies the **Dimension** being reported.
Value/ Dimension / LBound	Y	Y	Applies to array values. Specifies the lower bound of the array in the current dimension. On import, if this property is not specified, 0 is

			assumed.
Value/ Dimension/ NumberElements	Y	Y	Applies to array values. Specifies the number of elements in the current dimension. On import, this value must be specified, and it must be greater than 0.

Example

```
<Extension>
   <Name>VisioPageName</Name>
   <CopyWithParentID="0">False</CopyWithParent>
   <ValueIndex="1">
         <TypeID="8">BSTR</Type>
         <Data>NETWORK6:Subsystem1</Data>
   </Value>
</Extension>
```

Channels

Attributes

Attribute	**Applicable Values**	**Description/Notes**
Action	1 IGNORE 2 CREATE 3 CREATE_UNIQUE 4 UPDATE 5 MODIFY 6 DELETE 7 COMMISSION	To create a channel, you must specify the **Channel Name** and **TransceiverID** properties.

Properties

Property	**Supported by Function?**		**Description/Notes**
	Export	**Import**	
Name	Y	Y	The channel name is reported.
TransceiverId	Y	Y	An enumerated value indicating the channel type. The **ID** attribute is the transceiver ID as reported by LNS. The value is the transceiver name as determined in the StdXcvr.XML file. The possible values are as follows: 3 TP/XF-1250 1 TP/XF-78 5 TP/RS485-39 7 RF-10 9 PL-10 4 TP/FT-10 16 PL-20C 17 PL-20N 18 PL-30 12 TP/RS485-78 10 TP/RS485-625

			11 TP/RS485-1250 27 DC-78 28 DC-625 29 DC-1250 24 FO-20S 16 PL-20C-LOW 17 PL-20N-LOW 15 PL-20A 15 PL-20A-LOW 25 IP-10L 26 IP-10W 255 BACKPLANE 30 CUSTOM 152 FO-20L 142 DC33-78 143 DC33-625 144 DC33-1250
Delay	Y	Y	The expected longest round-trip time of a message traveling on the channel. You can specify the value of the **Delay** property, but the fact that the value has been overridden is not reported. If you want to use the default value, set this property to 0.

Example

```
<Channel Handle="1">
   <Name>Channel 1</Name>
   <TransceiverId ID="4">TP/FT-10</TransceiverId>
   <Delay>14</Delay>
</Channel>
```

Connection Descriptions

Attributes

Attribute	Applicable Values	Description/Notes
Action	1 IGNORE 2 CREATE 3 CREATE_UNIQUE 4 UPDATE 5 MODIFY 6 DELETE 7 COMMISSION	To create a connection description, you must specify the **Name** property.

Properties

Property	Supported by Function?		Description/Notes
	Export	Import	
Name	Y	Y	The connection description name is reported.
Description	Y	Y	The optional user-defined description is

			reported.
UseAuthenticationFlag	Y	Y	An enumerated value indicating whether the NV/MT connection uses authentication. The possible values are as follows: 0 False (does not use authentication) 1 True (uses authentication)
UsePriorityFlag	Y	Y	An enumerated value indicating whether the NV/MT connection uses priority messaging. The possible values are as follows: 0 False (does not use priority) 1 True (uses priority)
AliasOptions	Y	Y	An enumerated value indicating whether the NV/MT connection uses aliases to resolve selector conflicts. The possible values are as follows: 0 SelectorConflicts. Aliases are used automatically to resolve selector conflicts 1 Unicast. The LNS Object Server attempts to allocate an alias so that unicast addressing can be used if multicast addressing cannot (because of existing members or intersecting connections).
BroadcastOptions	Y	Y	An enumerated value indicating whether the NV/MT connection uses broadcast addressing. The possible values are as follows: 0 Never 1 Group 2 Always
PropertyOptions	Y	Y	An enumerated value indicating the default connection description properties (includes counts, timers, and service type) you have overridden. An instance of this property is reported for each option you have overridden. The possible values are as follows: 0 None 1 ServiceType 2 Priority 4 Auth 8 RetryCount 16 RepeatCount 32 RepeatTimer 64 ReceiveTimer 128 TransmitTimer 256 SuppressSourceAddr If you have not overridden any of these options, a single **PropertyOptions** property with a value

			of “None” is reported on export.
ReceiverTimer	Y	Y	An encoded integer. If you have not overridden the default receiver timer, this property is not reported on export. To use the LNS default value on import, delete this property.
RepeatCount	Y	Y	If you have not overridden the default repeat count, this property is not reported on export. To use the LNS default value on import, delete this property.
RepeatTimer	Y	Y	An encoded integer. If you have not overridden the default repeat timer, this property is not reported on export. To use the LNS default value on import, delete this property.
RetryCount	Y	Y	If you have not overridden the default retry count, this property is not reported on export. To use the LNS default value on import, delete this property.
ServiceType	Y	Y	An enumerated value indicating the message service type used by the connection. The possible values are as follows: 0 Ackd 1 UnackdRpt 2 Unackd 3 Request If you have not overridden the default service type, this property is not reported on export. To use the LNS default value on import, delete this property.
TransmitTimer	Y	Y	An encoded integer. If you have not overridden the default transmit timer, this property is not reported on export. To use the LNS default value on import, delete this property.

Example

```
<ConnectDescTemplate Handle="12">
   <Name>Repeated-Group Multicast</Name>
   <Description/>
   <UseAuthenticationFlag ID="0">False</UseAuthenticationFlag>
   <UsePriorityFlag ID="0">False</UsePriorityFlag>
   <AliasOptions ID="0">SelectorConflicts</AliasOptions>
   <BroadcastOptions ID="0">Never</BroadcastOptions>
   <PropertyOptions ID="1">ServiceType</PropertyOptions>
   <PropertyOptions ID="8">RetryCount</PropertyOptions>
   <PropertyOptions ID="16">RepeatCount</PropertyOptions>
   <PropertyOptions ID="32">RepeatTimer</PropertyOptions>
   <PropertyOptions ID="64">ReceiveTimer</PropertyOptions>
   <PropertyOptions ID="128">TransmitTimer</PropertyOptions>
   <ReceiveTimer>0</ReceiveTimer>
   <RepeatCount>3</RepeatCount>
   <RepeatTimer>1</RepeatTimer>
   <RetryCount>6</RetryCount>
   <ServiceType ID="1">UnackdRpt</ServiceType>
   <TransmitTimer>0</TransmitTimer>
</ConnectDescTemplate>
```

Device Templates

Attributes

Attribute	Applicable Values	Description/Notes
Action	1 IGNORE 2 CREATE 3 CREATE_UNIQUE 4 UPDATE 5 MODIFY 6 DELETE	Device templates will match existing entries based on the program ID specified in the XIF. To create a new device template, you must specify the **Name** and **XIF** properties.

Properties

Property	Supported by Function?		Description/Notes
	Export	Import	
Name	Y	Y	The name of the device template is reported.
XifPath	Y	Y	On export, this is the value reported by LNS. The XIF path may not always be identified or identified accurately. On import, if a device template is not defined, you must specify the **XifPath** because it is needed to locate and import the XIF. If the device template is defined, **XifPath** is not updated.
Classification/ ProgramID	Y	N	Device templates do not have handles; therefore, **Program ID** is used on import to match device templates in the XML file to the LNS network database.
Classification/ Format	Y	N	Specifies whether a device template has a LonMark standard program ID (8), a prototype ID (9), or a development ID (0).
Classification/ ManufacturerId	Y	N	Only reported if **Format** is 8 or 9.
Classification/ DeviceClass	Y	N	Only reported if **Format** is 8 or 9.
Classification/ DeviceSubclass	Y	N	Only reported if **Format** is 8 or 9.

Classification/ ModelNo	Y	N	Only reported if **Format** is 8 or 9.
DeviceValidation	Y	Y	An enumerated value indicating the level of validation performed by LNS when you commission an application device. The possible values are as follows: 0 Normal 1 NoChannelValidation 2 NoProgramInterfaceValidation 4 NoProgramIdValidation An instance of this property is reported on export for each validation option you have set. If you have not set any validation options, a single **DeviceValidation** property with a value of “None” is reported on export
SelfDocConsistency	Y	Y	An enumerated value indicating the required level of consistency for the self-documentation data (strings and format) of the application devices using the selected device template. The possible values are as follows: 0 IdenticalOnAllDevices 1 StringsMayDifferByDevice 2 StringsAndFormatMayDifferByDevice
ComponentApps	Y	Y	The set of LNS Plug-in applications registered for this device template at the device level are reported on export. The plug-ins for functional blocks are listed under the functional block elements.
FunctionalBlocks	Y	Y	See *Functional Blocks* for more information.
Extensions	Y	Y	See *Extensions* for more information.

Example

```
<DeviceTemplate>
   <Name>Echelon AI-10v3</Name>
   <XifPath>C:\LonWorks\Import\Echelon\LonPoint\Version3\AI-10v3</Xi
   fPath>
   <Classification>
         <ProgramId>80000105188A0403</ProgramId>
         <Format>8</Format>
         <ManufacturerId>000001</ManufacturerId>
         <DeviceClass>0518</DeviceClass>
         <DeviceSubclass>8A04</DeviceSubclass>
         <ModelNo>03</ModelNo>
   </Classification>
   <DeviceValidation ID="0">Normal</DeviceValidation>
   <SelfDocConsistency ID="1">StringsMayDifferByDevice
   </ComponentApps>
   </SelfDocConsistency>
   </FunctionalBlocks>
   </Extensions>
```

```
</DeviceTemplate>
```

Component Applications (Plug-Ins)

Component Application objects are reported on export at the system level (in which case they are reported at the same level as the **Channels**, **DeviceTemplates**, **ConnectDescTemplates**, and **RootSubsystem** elements), the **DeviceTemplates** level, and within the **FunctionalBlocks** property at the **DeviceTemplates** level.

You can register plug-ins on import; however, you need consider the following:

- You must define the Register command within the System Component Applications collection (defining the Deregister command in this collection is optional). This is because director applications use the Register command within this collection to determine whether the plug-in has already been registered. If the Register command is present in this collection, the plug-in is assumed to be registered. If the Register command is not defined, the plug-in is assumed to be not registered (even if other commands are specified for the plug-in elsewhere in the same network database).
- To delete a component application object, you must remove the Register command from the System Component Applications collection (as well as from all other locations). This enables director applications to determine if the plug-in is no longer registered.
- You can register the commands that the component application object implements at the system, device template, or device template/functional block level. The scope of the component application object determines at which level its commands are registered.
- Component application objects at the global level (within the LCA Object Server object) are not reported on exported, and you cannot register (add) or deregister (delete) them on import.

Attributes

Attribute	Applicable Values	Description/Notes
Action	1 IGNORE 2 CREATE 3 CREATE_UNIQUE 4 UPDATE 5 MODIFY 6 DELETE	To create a new plug-in, you must specify the **Name**, **RegisteredServerName**, and **ComponentClassID** properties.

Properties

Property	Supported by Function?		Description/Notes
	Export	Import	
Name	Y	Y	The name of the plug-in is reported.
RegisteredServer	Y	Y	Specifies the registered ActiveX name for the plug-in.
ComponentClassId	Y	Y	An enumerated value that specifies the type of object that the plug-in acts upon. The possible values are as follows: 1 Network 2 Networks 3 System

			4	Systems
			5	Subsystem
			6	Subsystems
			7	AppDevice
			8	AppDevices
			9	Router
			10	Routers
			11	RouterSide
			12	Channel
			13	Channels
			14	NetworkInterface
			15	NetworkInterfaces
			16	Subnet
			17	Subnets
			18	Connections
			19	Interface
			20	Interfaces
			21	TemplateLibrary
			22	NetworkVariable
			23	NetworkVariables
			24	MessageTag
			25	MessageTags
			26	ConfigProp
			27	ConfigProps
			28	LonMarkObject
			29	LonMarkObjects
			30	ComponentApp
			31	ComponentApps
			32	HardwareTemplate
			33	HardwareTemplates
			34	BuildTemplate
			35	BuildTemplates
			36	DeviceTemplate
			37	DeviceTemplates
			38	ProgramTemplate
			39	ProgramTemplates
			40	NetworkServiceDevice
			41	NetworkServiceDevices
			42	ConnectDescTemplate
			43	ConnectDescTemplates
			44	Error
			45	LonMarkAlarm
			46	ObjectStatus
			47	NetworkVariableField
			48	DetailInfo
			49	DataValue
			50	Extension
			51	Extensions
			52	RecoveryStatus
			53	CreditInfo
			54	Account
			55	Accounts
			56	BufferConfiguration
			57	FileTransfer
			58	Alias
			59	Aliases

			69 PingIntervals 70 Application 71 TestInfo 72 DataPoint 73 FormatSpec 74 MonitorSet 75 MonitorSets 76 MsgMonitorOptions 77 MsgMonitorPoint 78 MsgMonitorPoints 79 NvMonitorOptions 80 NvMonitorPoint 81 NvMonitorPoints 82 SourceAddress 83 LdrfLanguage 84 LdrfLanguages 85 ServiceStatus 86 UpgradeStatus 87 UpgradeInfo 88 UpgradeInfos 89 DatabaseValidationReport 90 DatabaseValidationErrorSummary 91 DatabaseValidationErrorSummaries 92 DatabaseValidationErrorInstance 93 NetworkResources 94 TypeSpec 95 FormatLocale 96 FormatLocales
DefaultAppFlag	Y	Y	If this flag is modified on import, you must update other conflicting plug-ins to clear the default flag.
CommandId	Y	Y	An enumerated value that specifies the operation performed by the plug-in. The possible values are as follows: 1 New 2 EditSource 10 BuildImage 11 Commission 12 Load 13 Configure 14 Calibrate 15 Connect 20 Browse 21 Monitor 22 Control 23 Report 24 Properties 30 Online 31 Offline 32 Reset 33 Test 34 Wink 35 Debug 40 Uninstall

			41 Replace 50 Register 51 Unregister 60 Recover 61 MonitorRecovery
VersionNumber	Y	Y	Specifies the plug-in version.
ManufacturerId	Y	Y	Specifies the plug-in manufacturer.

Example

```
<ComponentApp>
   <Name>Echelon LonPoint Configuration</Name>
   <RegisteredServer>EchelonLonPointConfiguration.Application</Regis
   teredServer>
   <ComponentClassId ID="3">System</ComponentClassId>
   <Description>Echelon LonPoint Configuration</Description>
   <DefaultAppFlag ID="0">False</DefaultAppFlag>
   <CommandId ID="50">Register</CommandId>
   <VersionNumber>3.10</VersionNumber>
   <ManufacturerId>Echelon Corporation</ManufacturerId>
</ComponentApp>
```

Programming Example

The following programming example, written in Visual C# with Microsoft Visual Studio 2008, demonstrates how to programmatically export a LonMaker drawing, create a LonMaker network with XML, and then import the XML file into the LonMaker tool. The network created by this example contains one device that is commissioned, two functional blocks, one network variable in each functional block, and one connection that binds the network variables.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading;
using System.Xml;
using PlugInWrapper; //reference to Echelon PluginWrapper.dll file.
                     //You can use download and reference this assembly instead of
                     //creating your own director application.

namespace myLmXmlNetwork
{
    class myLmXmlNetwork
    {
        static void Main(string[] args)
        {
            //Create an instance of the LonMaker plug-in using its registered ActiveX name,
            //which is "EchelonLonMakerXML.Application"
            LonMakerXmlPlugIn m_lmXml = new LonMakerXmlPlugIn();

            //Set the network name property to open the network
            m_lmXml.NetworkName = "MyNetwork";

            // **Insert export automation properties before invoking the send command
            //Make plug-in visible so it shows itself
            m_lmXml.Visible = true;

            //Export LonMaker drawing to XML file
            //Invoke the send command to specify the action and target
            //object.  The send command uses the following syntax:
            //(23, 5, “network/system/subsystem[/subsystem…])”

            m_lmXml.SendCommand(23, 5, "MyNetwork/MyNetwork/Subsystem 1");
```

```
//create Directory for XML file
DirectoryInfo di = Directory.CreateDirectory("C:\\Lm\\XML\\MyNetwork\\");

//create XML file
m_lmXml.XmlFileName = "C:\\Lm\\XML\\MyNetwork\\Export.XML";

// Create the XML writer options
XmlWriterSettings settings = new XmlWriterSettings();
settings.ConformanceLevel = ConformanceLevel.Document;
settings.Indent = true;
settings.IndentChars = ("    ");

//Create the XML Writer
using (XmlWriter writer = XmlWriter.Create(m_lmXml.XmlFileName))
{
    writer.WriteProcessingInstruction("xml-stylesheet", "type=\"text/xsl\"
        href='file://" + "C:\\LonWorks\\LonMaker\\XML\\LonMaker.xsl" +
        "'");

    // Start Network Elemeny
    writer.WriteStartElement("LonWorksNetwork");
    writer.WriteElementString("Name", "MyNetwork");

    // Specify root subsystem
    writer.WriteElementString("RootSubsystem", "");

    // Write the domain ID
    // Use random 3-byte domain ID; first sleep a ms to advance timer
    writer.WriteStartElement("DomainId");
    Random randobj = new Random();
    writer.WriteString(String.Format("{0:X2}", randobj.Next(255)));
    writer.WriteString(String.Format("{0:X2}", randobj.Next(255)));
    writer.WriteString(String.Format("{0:X2}", randobj.Next(255)));
    writer.WriteEndElement();

    // Write the export scope
    writer.WriteStartElement("ExportScope");
    writer.WriteAttributeString("ID", "0");
    writer.WriteString("ALL");
    writer.WriteEndElement();

    // Start the Top-level Subsystems Elements
    writer.WriteStartElement("Subsystems");

    // Start the Subsystem Element
    writer.WriteStartElement("Subsystem");
    writer.WriteElementString("Name", "Subsystem 1");

    //Start AppDevices Elements
    writer.WriteStartElement("AppDevices");

    //Write AppDevice Element
    writer.WriteStartElement("AppDevice");
    writer.WriteAttributeString("Action", "CREATE");
    writer.WriteAttributeString("Commission", "TRUE");
    writer.WriteElementString("Name", "DIO-1");
    writer.WriteElementString("Channel", "Channel 1");
    writer.WriteElementString("Template", "Echelon DIO-10v3");
    writer.WriteElementString("NeuronId", "00A145784600");
    writer.WriteElementString("Image",
        "C:\\LonWorks\\Import\\Echelon\\LonPoint\\Version3\\
         DIO-10v3.apb");

    // Start FunctionalBlock Elements
    writer.WriteStartElement("FunctionalBlocks");

    // Write FunctionalBlock Element#1
    writer.WriteStartElement("FunctionalBlock");
    writer.WriteAttributeString("Handle", "5");
    writer.WriteElementString("Name", "DE-1");
    writer.WriteStartElement("IsVirtualFb");
    writer.WriteAttributeString("ID", "0");
    writer.WriteString("False");
    writer.WriteEndElement();

    writer.WriteElementString("ProgrammaticName", "Digital Encoder[0]");
    writer.WriteElementString("FuncProfileName", "Echelon Digital Encoder");

    writer.WriteStartElement("Scope");
    writer.WriteAttributeString("ID", "3");
    writer.WriteString("Manf");
    writer.WriteEndElement();

    writer.WriteElementString("FuncProfileProgrammaticName", "UFPTDigitalEncoder");
    writer.WriteElementString("SubsystemName", "$Subsystem 1");

    writer.WriteStartElement("IsDynamic");
    writer.WriteAttributeString("ID", "0");
    writer.WriteString("False");
    writer.WriteEndElement();

    // Start NetworkVariables Element
    writer.WriteStartElement("NetworkVariables");

    // Write NetworkVariable Element
    writer.WriteStartElement("NetworkVariable");
    writer.WriteAttributeString("Handle", "5");
    writer.WriteElementString("Name", "Digital_Out");
    writer.WriteElementString("ProgrammaticName", "DE_D_Out_1");
    writer.WriteElementString("LonMarkMemberNumber", "6");

    writer.WriteStartElement("Direction");
    writer.WriteAttributeString("ID", "1");
    writer.WriteString("OUTPUT");
    writer.WriteEndElement();

    writer.WriteStartElement("TypeSpec");
    writer.WriteStartElement("Scope");
    writer.WriteAttributeString("ID", "0");
    writer.WriteString("Standard");
    writer.WriteEndElement();
    writer.WriteElementString("TypeName", "SNVT_switch");
    writer.WriteEndElement();

    writer.WriteElementString("Format", "SNVT_switch");

    writer.WriteStartElement("HasShape");
    writer.WriteAttributeString("ID", "1");
    writer.WriteString("True");
    writer.WriteEndElement();

    writer.WriteStartElement("IsDynamic");
    writer.WriteAttributeString("ID", "0");
    writer.WriteString("False");
    writer.WriteEndElement();

    // Start Network Variable Connection Targets(Target Elements)
    writer.WriteStartElement("Targets");

    // Write Network Variable Connection (Target Element)
    writer.WriteStartElement("Target");
    writer.WriteElementString("SubsystemName", "$Subsystem 1");
    writer.WriteElementString("DeviceName", "DIO-1");
    writer.WriteElementString("FunctionalBlockName", "Digital Output[0]");
    writer.WriteElementString("TargetName", "DO_Digital");

    // End Target Element
    writer.WriteEndElement();

    // End Targets Element
```

```
writer.WriteEndElement();

// End NetworkVariable Element
writer.WriteEndElement();

// End NetworkVariables Element
writer.WriteEndElement();

// End FunctionalBlock Element
writer.WriteEndElement();

// Write functional block #2
writer.WriteStartElement("FunctionalBlock");
writer.WriteAttributeString("Handle", "3");
writer.WriteElementString("Name", "Digital Output[0]");

writer.WriteStartElement("IsVirtualFb");
writer.WriteAttributeString("ID", "0");
writer.WriteString("False");
writer.WriteEndElement();

writer.WriteElementString("ProgrammaticName", "Digital Output[0]]");
writer.WriteElementString("FuncProfileName", "Echelon Digital Output");

writer.WriteStartElement("Scope");
writer.WriteAttributeString("ID", "3");
writer.WriteString("Manf");
writer.WriteEndElement();

writer.WriteElementString("FuncProfileProgrammaticName", "UFPTDigitalOutput");
writer.WriteElementString("SubsystemName", "$Subsystem 1");

writer.WriteStartElement("IsDynamic");
writer.WriteAttributeString("ID", "0");
writer.WriteString("False");
writer.WriteEndElement();

// Start NetworkVariables Element
writer.WriteStartElement("NetworkVariables");

// Write network variable
writer.WriteStartElement("NetworkVariable");
writer.WriteAttributeString("Handle", "0");
writer.WriteElementString("Name", "DO_Digital");
writer.WriteElementString("ProgrammaticName", "DO_Digital_1");
writer.WriteElementString("LonMarkMemberNumber", "0");

writer.WriteStartElement("Direction");
writer.WriteAttributeString("ID", "0");
writer.WriteString("INPUT");
writer.WriteEndElement();

writer.WriteStartElement("TypeSpec");
writer.WriteStartElement("Scope");
writer.WriteAttributeString("ID", "0");
writer.WriteString("Standard");
writer.WriteEndElement();
writer.WriteElementString("TypeName", "SNVT_switch");
writer.WriteEndElement();

writer.WriteElementString("Format", "SNVT_switch");

writer.WriteStartElement("HasShape");
writer.WriteAttributeString("ID", "1");
writer.WriteString("True");
writer.WriteEndElement();

writer.WriteStartElement("IsDynamic");
writer.WriteAttributeString("ID", "0");
writer.WriteString("False");
writer.WriteEndElement();
```

```
    // End NetworkVariable Element
    writer.WriteEndElement();

    // End NetworkVariables Element
    writer.WriteEndElement();

    // End FunctionalBlock Element
    writer.WriteEndElement();

    // End FunctionalBlocks Element
    writer.WriteEndElement();

    //End AppDevice Element
    writer.WriteEndElement();

    //End AppDevices Element
    writer.WriteEndElement();

    //End Subsystem Element
    writer.WriteEndElement();

    //End Top-level Subsystem

    writer.WriteEndElement();

    //Start Channels
    writer.WriteStartElement("Channels");

    //Write Channel
    writer.WriteStartElement("Channel");
    writer.WriteElementString("Name", "Channel 1");
    writer.WriteStartElement("TransceiverId");
    writer.WriteAttributeString("ID", "4");
    writer.WriteString("TP/FT-10");
    writer.WriteEndElement();

    //End Channel
    writer.WriteEndElement();

    //End Channels
    writer.WriteEndElement();

    // Start Device Templates
    writer.WriteStartElement("DeviceTemplates");

    //Write Device Template
    writer.WriteStartElement("DeviceTemplate");
    writer.WriteElementString("Name", "Echelon DIO-10v3");
    writer.WriteElementString("XifPath",
               "C:\\LonWorks\\Import\\Echelon\\LonPoint\\Version3\\
                DIO-10v3");
    //end device tameplates
    writer.WriteEndElement();

    //end device tameplate
    writer.WriteEndElement();

    //end network
    writer.WriteEndElement();
}

 //Import XML file into LonMaker tool
 //Invoke the send command to specify the action and target object.
 //The send command uses the following syntax:
//(10000, 5, “network/system/root subsystem[/subsystem…])”
m_lmXml.SendCommand(10000, 5, "MyNetwork/MyNetwork/Subsystem 1");

//show the export plug-in and import plug-in dialogs
 m_lmXml.Visible = true;
```

```
            //Option to create and specify location of import log
            m_lmXml.LogFileName = "C:\\Lm\\XML\\MyNetwork\\Import.log";
        }
    }
}
```

Appendix A

LonWorks Network XML Schema

This appendix presents the XML schema that defines the structure and content of a LonMaker network XML file.

LONWORKS XML Schema

The XML schema used for LonWorks network is as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://www.echelon.com/LonMaker"
  elementFormDefault="qualified"
  xmlns="http://www.echelon.com/LonMaker"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="LonWorksNetwork">
   <xs:complexType>
    <xs:sequence>
        <xs:element name="Name" type="xs:string" minOccurs="0" />
        <xs:element name="ReportCreated" type="xs:dateTime" minOccurs="0" />
        <xs:element name="RootSubsystem" type="xs:string" minOccurs="0" />
        <xs:element name="DomainId" type="xs:string" minOccurs="0" />
        <xs:element name="ExportScope" type="enumType" minOccurs="0" />
        <xs:element name="Comment" type="xs:string" minOccurs="0" maxOccurs="1" />
        <xs:element name="Subsystems" type="subsysCollection" minOccurs="0" />
        <xs:element name="Channels" type="chanCollection" minOccurs="0" maxOccurs="1" />
        <xs:element name="DeviceTemplates" type="devTmplCollection" minOccurs="0"
        maxOccurs="1" />
        <xs:element name="ConnectDescTemplates" type="cdCollection" minOccurs="0"
        maxOccurs="1" />
    </xs:sequence>
   </xs:complexType>
  </xs:element>

  <xs:complexType name="enumType">
   <xs:complexContent>
    <xs:extension base="xs:anyType">
     <xs:attribute name="ID" type="xs:integer" use="optional" />
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="objectType">
   <xs:sequence />
   <xs:attribute name="Handle" type="xs:integer" use="optional" />
   <xs:attribute name="Action" type="xs:string" use="optional" />
  </xs:complexType>

  <xs:complexType name="valueType">
   <xs:complexContent>
    <xs:extension base="xs:anyType">
     <xs:attribute name="Index" type="xs:integer" />
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="extenValue">
   <xs:sequence>
    <xs:element name="Type" type="enumType" maxOccurs="1" minOccurs="0" />
    <xs:element name="Data" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="NumberDimensions" type="xs:integer" maxOccurs="1"
    minOccurs="0" />
    <xs:element name="ElementSize" type="xs:integer" maxOccurs="1" minOccurs="0" />
    <xs:element name="Dimension" type="extenDimension" maxOccurs="1" minOccurs="0" />
   </xs:sequence>
   <xs:attribute name="Index" type="xs:integer" />
  </xs:complexType>

  <xs:complexType name="extenDimension">
   <xs:sequence>
    <xs:element name="LBound" type="xs:integer" maxOccurs="1" minOccurs="0" />
    <xs:element name="NumberElements" type="xs:integer" />
   </xs:sequence>
   <xs:attribute name="Index" type="xs:integer" />
  </xs:complexType>

  <xs:complexType name="extenObject">
   <xs:sequence>
    <xs:element name="Name" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="CopyWithParent" type="enumType" maxOccurs="1" minOccurs="0" />
    <xs:element name="Value" type="extenValue" maxOccurs="3" minOccurs="0" />
   </xs:sequence>
  </xs:complexType>

  <xs:complexType name="extenCollection">
   <xs:sequence>
    <xs:element name="Extension" type="extenObject" maxOccurs="unbounded"
    minOccurs="0" />
   </xs:sequence>
  </xs:complexType>

  <xs:complexType name="cpObject">
   <xs:complexContent>
    <xs:extension base="objectType">
     <xs:sequence>
      <xs:element name="Name" type="xs:string" maxOccurs="1" minOccurs="0" />
      <xs:element name="DeviceSpecificAttribute" type="enumType" maxOccurs="1"
       minOccurs="0" />
      <xs:element name="ConstantAttribute" type="enumType" maxOccurs="1"
       minOccurs="0" />
      <xs:element name="Dimension" type="xs:integer" maxOccurs="1" minOccurs="0" />
      <xs:element name="FlagsByte" type="enumType" maxOccurs="16" minOccurs="0" />
      <xs:element name="Value" type="valueType" maxOccurs="unbounded"
       minOccurs="0" />
     </xs:sequence>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="cpCollection">
   <xs:sequence>
    <xs:element name="ConfigProperty" type="cpObject" maxOccurs="unbounded"
    minOccurs="0" />
   </xs:sequence>
  </xs:complexType>

  <xs:complexType name="typeSpecType">
   <xs:sequence>
    <xs:element name="Scope" type="enumType" maxOccurs="1" minOccurs="0" />
    <xs:element name="ProgramId" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="TypeName" type="xs:string" maxOccurs="1" minOccurs="0" />
   </xs:sequence>
  </xs:complexType>

  <xs:complexType name="nvObject">
   <xs:complexContent>
    <xs:extension base="objectType">
     <xs:sequence>
      <xs:element name="Name" type="xs:string" maxOccurs="1" minOccurs="0" />
      <xs:element name="Index" type="xs:integer" maxOccurs="1" minOccurs="0" />
      <xs:element name="LonMarkMemberIndex" type="xs:integer" maxOccurs="1"
       minOccurs="0" />
      <xs:element name="ProgrammaticName" type="xs:string" maxOccurs="1"
       minOccurs="0" />
      <xs:element name="LonMarkMemberNumber" type="xs:integer" maxOccurs="1"
       minOccurs="0" />
      <xs:element name="Direction" type="enumType" maxOccurs="1" minOccurs="0" />
      <xs:element name="IsTypeConfigurable" type="enumType" maxOccurs="1"
       minOccurs="0" />
      <xs:element name="TypeSpec" type="typeSpecType" maxOccurs="1" minOccurs="0" />
      <xs:element name="Format" type="xs:string" maxOccurs="1" minOccurs="0" />
      <xs:element name="IsDynamic" type="enumType" maxOccurs="1" minOccurs="0" />
      <xs:element name="Selector" type="xs:integer" maxOccurs="1" minOccurs="0" />
      <xs:element name="Value" type="xs:string" maxOccurs="1" minOccurs="0" />
      <xs:element name="HasShape" type="enumType" maxOccurs="1" minOccurs="0" />
      <xs:element name="ConnDesc" type="xs:string" maxOccurs="1" minOccurs="0" />
      <xs:element name="ConfigProperties" type="cpCollection" maxOccurs="1"
       minOccurs="0" />
      <xs:element name="Targets" type="targetCollection" maxOccurs="1"
       minOccurs="0" />
     </xs:sequence>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="targetObject">
   <xs:sequence>
    <xs:element name="ReferenceName" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="SubsystemName" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="DeviceName" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="FunctionalBlockName" type="xs:string" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="TargetName" type="xs:string" maxOccurs="1" minOccurs="0" />
   </xs:sequence>
  </xs:complexType>

  <xs:complexType name="targetCollection">
   <xs:sequence>
    <xs:element name="Target" type="targetObject" maxOccurs="unbounded"
     minOccurs="0" />
   </xs:sequence>
  </xs:complexType>

  <xs:complexType name="mtObject">
   <xs:complexContent>
    <xs:extension base="objectType">
```

```
    <xs:sequence>
     <xs:element name="Name" type="xs:string" maxOccurs="1" minOccurs="0" />
     <xs:element name="Direction" type="enumType" maxOccurs="1" minOccurs="0" />
     <xs:element name="AddressTableIndex" type="xs:integer" maxOccurs="1"
      minOccurs="0" />
     <xs:element name="ConnDesc" type="xs:string" maxOccurs="1" minOccurs="0" />
     <xs:element name="HasShape" type="enumType" maxOccurs="1" minOccurs="0" />
     <xs:element name="IsDynamic" type="enumType" maxOccurs="1" minOccurs="0" />
     <xs:element name="Targets" type="targetCollection" maxOccurs="1"
      minOccurs="0" />
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="nvCollection">
  <xs:sequence>
   <xs:element name="NetworkVariable" type="nvObject" maxOccurs="unbounded"
     minOccurs="0" />
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="mtCollection">
  <xs:sequence>
   <xs:element name="MessageTag" type="mtObject" maxOccurs="unbounded"
     minOccurs="0" />
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="fbObject">
  <xs:complexContent>
   <xs:extension base="objectType">
    <xs:sequence>
     <xs:element name="Name" type="xs:string" maxOccurs="1" minOccurs="0" />
     <xs:element name="IsVirtualFb" type="enumType" maxOccurs="1" minOccurs="0" />
     <xs:element name="ProgrammaticName" type="xs:string" maxOccurs="1"
      minOccurs="0" />
     <xs:element name="FuncProfileName" type="xs:string" maxOccurs="1"
      minOccurs="0" />
     <xs:element name="Scope" type="enumType" maxOccurs="1" minOccurs="0" />
     <xs:element name="FuncProfileProgrammaticName" type="xs:string"
      maxOccurs="1" minOccurs="0" />
     <xs:element name="FuncProfileDescription" type="xs:string" maxOccurs="1"
      minOccurs="0" />
     <xs:element name="PrincipalNv" type="xs:string" maxOccurs="1" minOccurs="0"
      />
     <xs:element name="ComponentApps" type="compAppCollection" maxOccurs="1"
      minOccurs="0" />
     <xs:element name="Shape" type="shapeType" maxOccurs="1" minOccurs="0" />
     <xs:element name="IsDynamic" type="enumType" maxOccurs="1" minOccurs="0" />
     <xs:element name="SubsystemName" type="xs:string" maxOccurs="1"
      minOccurs="0" />
     <xs:element name="Extensions" type="extenCollection" maxOccurs="1"
      minOccurs="0" />
     <xs:element name="MessageTags" type="mtCollection" maxOccurs="1"
      minOccurs="0" />
     <xs:element name="NetworkVariables" type="nvCollection" maxOccurs="1"
      minOccurs="0" />
     <xs:element name="ConfigProperties" type="cpCollection" maxOccurs="1"
      minOccurs="0" />
    </xs:sequence>
    <xs:attribute name="Ref" type="xs:string" />
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="shapeType">
  <xs:sequence>
   <xs:element name="StencilName" type="xs:string" maxOccurs="1" minOccurs="0" />
   <xs:element name="MasterName" type="xs:string" maxOccurs="1" minOccurs="0" />
   <xs:element name="PinX" type="xs:float" maxOccurs="1" minOccurs="0" />
   <xs:element name="PinY" type="xs:float" maxOccurs="1" minOccurs="0" />
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="compAppObject">
  <xs:complexContent>
   <xs:extension base="objectType">
    <xs:sequence>
     <xs:element name="Name" type="xs:string" maxOccurs="1" minOccurs="0" />
     <xs:element name="RegisteredServer" type="xs:string" maxOccurs="1"
      minOccurs="0" />
     <xs:element name="ComponentClassId" type="enumType" maxOccurs="1"
      minOccurs="0" />
     <xs:element name="Description" type="xs:string" maxOccurs="1" minOccurs="0" />
     <xs:element name="DefaultAppFlag" type="enumType" maxOccurs="1"
      minOccurs="0" />
     <xs:element name="CommandId" type="enumType" maxOccurs="1" minOccurs="0" />
     <xs:element name="VersionNumber" type="xs:string" maxOccurs="1"
      minOccurs="0" />
     <xs:element name="ManufacturerId" type="xs:string" maxOccurs="1"
      minOccurs="0" />
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="compAppCollection">
  <xs:sequence>
   <xs:element name="ComponentApp" type="compAppObject" maxOccurs="unbounded"
     minOccurs="0" />
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="simpleObjType">
  <xs:sequence />
  <xs:attribute name="Handle" type="xs:integer" />
 </xs:complexType>

 <xs:complexType name="subnetObject">
  <xs:complexContent>
   <xs:extension base="simpleObjType">
    <xs:sequence>
```

```
    <xs:element name="Name" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="NodeId" type="xs:integer" maxOccurs="1" minOccurs="0" />
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="deviceObject">
 <xs:complexContent>
  <xs:extension base="comObjectType">
   <xs:sequence>
    <xs:element name="NeuronId" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="Name" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="LocationText" type="xs:string" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="Location" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="Template" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="Channel" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="Description" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="SelfDocumentation" type="xs:string" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="AuthenticationEnabled" type="enumType" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="InitialAuthenticationKey" type="xs:string" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="Priority" type="xs:integer" maxOccurs="1" minOccurs="0" />
    <xs:element name="AliasCapacity" type="xs:integer" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="AliasUseCount" type="xs:integer" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="AddressCapacity" type="xs:integer" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="AddressUseCount" type="xs:integer" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="Shape" type="shapeType" maxOccurs="1" minOccurs="0" />
    <xs:element name="CommissionStatus" type="enumType" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="AttachmentStatus" type="enumType" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="PingClass" type="enumType" maxOccurs="1" minOccurs="0" />
    <xs:element name="Subnet" type="subnetObject" maxOccurs="1" minOccurs="0" />
    <xs:element name="Extensions" type="extenCollection" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="FunctionalBlocks" type="fbCollection" maxOccurs="1"
     minOccurs="0" />
   </xs:sequence>
  <xs:attribute name="Ref" type="xs:string" />
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="fbCollection">
 <xs:sequence>
  <xs:element name="FunctionalBlock" type="fbObject" maxOccurs="unbounded"
   minOccurs="0" />
 </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="deviceCollection">
 <xs:sequence>
  <xs:element name="AppDevice" type="deviceObject" maxOccurs="unbounded"
   minOccurs="0" />
 </xs:sequence>
</xs:complexType>

<xs:complexType name="routerObject">
 <xs:complexContent>
  <xs:extension base="comObjectType">
   <xs:sequence>
    <xs:element name="Name" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="LocationText" type="xs:string" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="Location" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="Description" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="AuthenticationEnabled" type="enumType" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="InitialAuthenticationKey" type="xs:string" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="Shape" type="shapeType" maxOccurs="1" minOccurs="0" />
    <xs:element name="Class" type="enumType" maxOccurs="1"
     minOccurs="0"></xs:element>
    <xs:element name="PingClass" type="enumType" maxOccurs="1" minOccurs="0" />
    <xs:element name="CommissionStatus" type="enumType" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="AttachmentStatus" type="enumType" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="Extensions" type="extenCollection" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="NearRouterSide" type="routerSide" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="FarRouterSide" type="routerSide" maxOccurs="1"
     minOccurs="0" />
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="routerSide">
 <xs:sequence>
  <xs:element name="NeuronId" type="xs:string" maxOccurs="1" minOccurs="0" />
  <xs:element name="Channel" type="xs:string" maxOccurs="1" minOccurs="0" />
  <xs:element name="Priority" type="xs:integer" maxOccurs="1" minOccurs="0" />
  <xs:element name="Subnet" type="subnetObject" maxOccurs="1" minOccurs="0" />
 </xs:sequence>
</xs:complexType>

<xs:complexType name="routerCollection">
 <xs:sequence>
  <xs:element name="Router" type="routerObject" maxOccurs="unbounded"
   minOccurs="0" />
 </xs:sequence>
</xs:complexType>

<xs:complexType name="subsysObject">
```

```
 <xs:complexContent>
  <xs:extension base="objectType">
   <xs:sequence>
    <xs:element name="Name" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="DocumentName" type="xs:string" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="PageName" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="Location" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="SubsystemId" type="xs:integer" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="Shape" type="shapeType" maxOccurs="1" minOccurs="0" />
    <xs:element name="Description" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="Extensions" type="extenCollection" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="Routers" type="routerCollection" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="AppDevices" type="deviceCollection" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="Subsystems" type="subsysCollection" maxOccurs="1"
     minOccurs="0" />
   </xs:sequence>
  <xs:attribute name="Ref" type="xs:string" />
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="subsysCollection">
 <xs:sequence>
  <xs:element name="Subsystem" type="subsysObject" maxOccurs="unbounded"
    minOccurs="0" />
 </xs:sequence>
</xs:complexType>

<xs:complexType name="chanObject">
 <xs:complexContent>
  <xs:extension base="objectType">
   <xs:sequence>
    <xs:element name="Name" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="Delay" type="xs:integer" maxOccurs="1" minOccurs="0" />
    <xs:element name="TransceiverId" type="enumType" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="Extensions" type="extenCollection" maxOccurs="1"
     minOccurs="0" />
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="chanCollection">
 <xs:sequence>
  <xs:element name="Channel" type="chanObject" maxOccurs="unbounded"
    minOccurs="0" />
 </xs:sequence>
</xs:complexType>

<xs:complexType name="devTmplObject">
 <xs:complexContent>
```

```
  <xs:extension base="objectType">
   <xs:sequence>
    <xs:element name="Name" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="XifPath" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="Classification" type="dtClassType" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="DeviceValidation" type="enumType" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="SelfDocConsistency" type="enumType" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="ComponentApps" type="compAppCollection" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="FunctionalBlocks" type="fbCollection" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="Extensions" type="extenCollection" maxOccurs="1"
     minOccurs="0" />
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="dtClassType">
 <xs:sequence>
  <xs:element name="ProgramId" type="xs:string" maxOccurs="1" minOccurs="0" />
  <xs:element name="Format" type="xs:integer" maxOccurs="1" minOccurs="0" />
  <xs:element name="ManufacturerId" type="xs:string" maxOccurs="1"
   minOccurs="0" />
  <xs:element name="DeviceClass" type="xs:string" maxOccurs="1" minOccurs="0" />
  <xs:element name="DeviceSubclass" type="xs:string" maxOccurs="1"
   minOccurs="0" />
  <xs:element name="ModelNo" type="xs:string" maxOccurs="1" minOccurs="0" />
 </xs:sequence>
</xs:complexType>

<xs:complexType name="devTmplCollection">
 <xs:sequence>
  <xs:element name="DeviceTemplate" type="devTmplObject" maxOccurs="unbounded"
   minOccurs="0" />
 </xs:sequence>
</xs:complexType>

<xs:complexType name="cdObject">
 <xs:complexContent>
  <xs:extension base="objectType">
   <xs:sequence>
    <xs:element name="Name" type="xs:string" minOccurs="0" />
    <xs:element name="Description" type="xs:string" maxOccurs="1" minOccurs="0" />
    <xs:element name="UseAuthenticationFlag" type="enumType" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="UsePriorityFlag" type="enumType" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="AliasOptions" type="enumType" maxOccurs="1" minOccurs="0" />
    <xs:element name="BroadcastOptions" type="enumType" maxOccurs="1"
     minOccurs="0" />
    <xs:element name="PropertyOptions" type="enumType" minOccurs="0"
     maxOccurs="16" />
```

```
      <xs:element name="ReceiveTimer" type="xs:integer" maxOccurs="1"
       minOccurs="0" />
      <xs:element name="RepeatCount" type="xs:integer" maxOccurs="1"
       minOccurs="0" />
      <xs:element name="RepeatTimer" type="xs:integer" maxOccurs="1"
       minOccurs="0" />
      <xs:element name="RetryCount" type="xs:integer" maxOccurs="1" minOccurs="0" />
      <xs:element name="TransmitTimer" type="xs:integer" maxOccurs="1"
       minOccurs="0" />
      <xs:element name="ServiceType" type="enumType" maxOccurs="1" minOccurs="0" />
     </xs:sequence>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="cdCollection">
   <xs:sequence>
    <xs:element name="ConnectDescTemplate" type="cdObject" maxOccurs="unbounded"
     minOccurs="0" />
   </xs:sequence>
  </xs:complexType>

  <xs:complexType name="comObjectType">
   <xs:complexContent>
    <xs:extension base="objectType">
     <xs:sequence />
      <xs:attribute name="Commission" type="xs:string" />
      <xs:attribute name="State" type="xs:string" />
      <xs:attribute name="CP" type="xs:string" />
      <xs:attribute name="DevCP" type="xs:string" />
      <xs:attribute name="Image" type="xs:string" />
      <xs:attribute name="FwUpdate" type="xs:string" />
     </xs:extension>
    </xs:complexContent>
  </xs:complexType>

</xs:schema>
```

ECHELON®

www.echelon.com

078-0466-01A